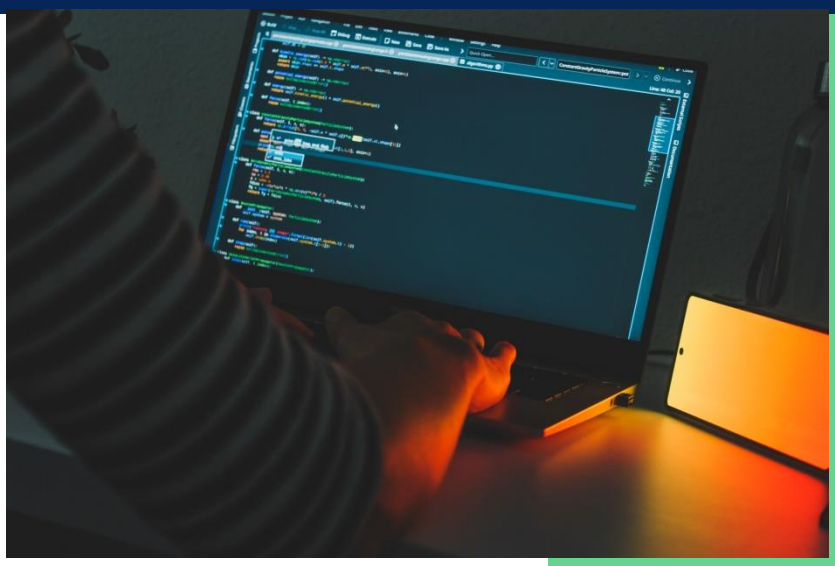
Съфинансиран от програма
„Еразъм+“
на Европейския съюз

Посетете нашата уеб страница
https://makers-project.eu

Въведение в Python



Година на публикуване: **2022**

Редактор: **Гергана Цисарова-Димитрова**

Проект “Училища за творци: Включване на 3D дизайн и програмиране в обучението в средните училища с цел насърчаване креативността и ангажираността на учениците с науката, технологиите, инженерството и математиката” (договор да безвъзмездна помощ №2020-1-BG01-KA201-079274)

MAKER SCHOOLS
3D Design for Education

Съдържание

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

Защо да учите Python?

Python е език за програмиране, създаден през 1991 г. Той е популярен и лесен за изучаване и в момента е много търсен на пазара на труда. Популярността му нараства, тъй като се използва широко в науката за данните, изкуствения интелект и машинното обучение. Ако желаете в бъдеще да работите в областта на изкуствения интелект и машинното обучение, изучаването на Python е задължително за вас. Python е универсален език и може да се използва както за прости, така и за много сложни задачи. Той може да се използва за разработване на back-end частта на уеб приложение (на сървър), но не може да се използва за разработване на front-end (това, което потребителят вижда и с което взаимодейства в браузъра). Можете да използвате Python за свързване към бази данни. Python работи на различни платформи (Windows, Mac, Linux, Raspberry Pi и др.).

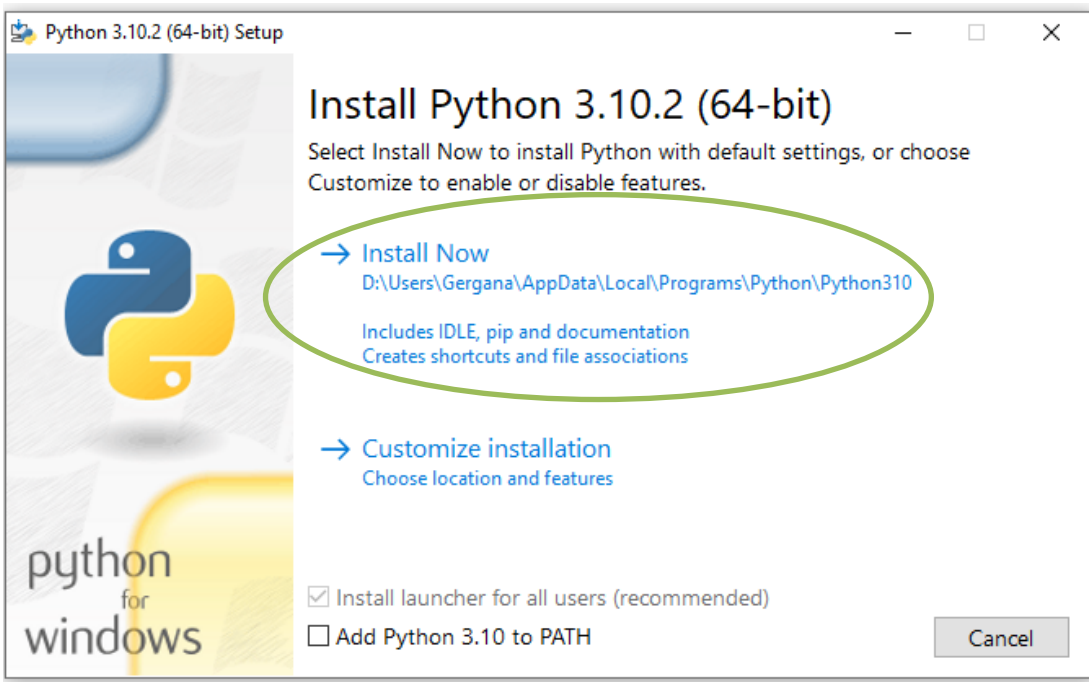
Започнете работа с Python

Инсталирайте интегрирана среда за разработка (ИСР)

Преди да започнете да изучавате Python, първо трябва да инсталирате самия Python, ако на компютъра ви още не е инсталиран (на повечето компютри с Windows не е).

Изтеглете последната версия, подходяща за вашата операционна система, от https://www.python.org/downloads/. Ако искате директно да изтеглите версията за Windows от януари 2022 г., щракнете тук.

Стартирайте стандартната инсталация, както е показано на екрана по-долу, и следвайте инструкциите.



Python може да се стартира дори от командната конзола в Windows (Command Line, CMD). Това обаче няма да е приятно преживяване. За да работите ефективно с Python, се препоръчва да изберете и инсталирате интегрирана среда за разработка (ИСР) - специализиран софтуер, който ви позволява да компилирате и стартирате скриптове на Python, да създавате и запаметявате файлове и проекти и т.н. Съществуват много варианти за ИСР, които лесно можете да намерите в интернет. Изборът на ИСР обикновено е

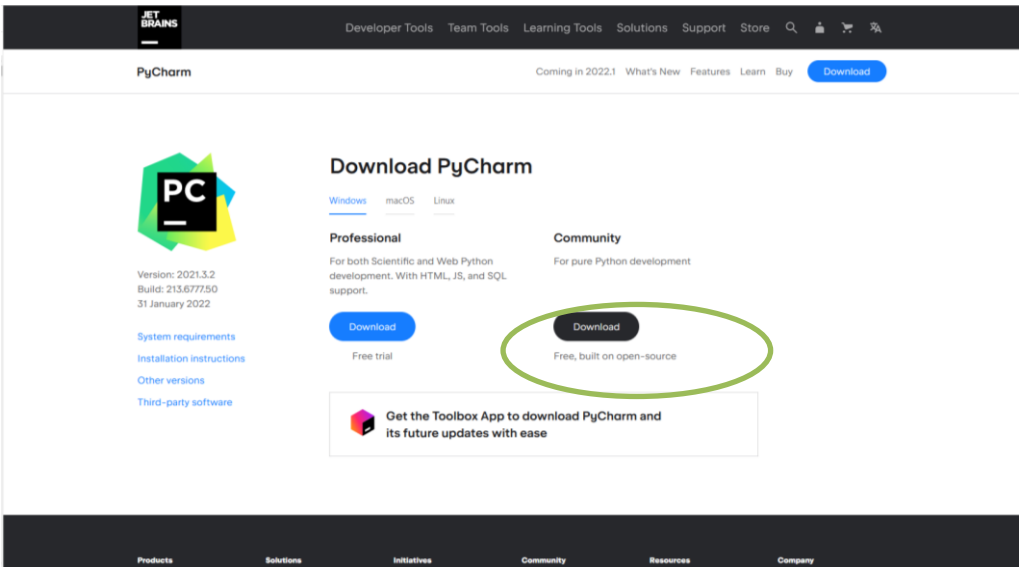
Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

въпрос на лични предпочитания. Ако вече използвате някоя популярна ИСР за друг език (напр. Visual Studio Code или някоя ИСР на JetBrains), вероятно вече сте свикнали с нейния интерфейс и е добра идея да използвате същата за Python, ако се поддържа. За изучаване на Pyhton препоръчваме да използвате Visual Studio Code или PyCharm Community (Community е безплатната версия, а има и платена версия с повече функции).

PyCharm може да бъде от голяма полза за начинаещите, тъй като подсказва корекции на типични грешки и има мощни функции за подсказване на код, докато пишете. Недостатъците са, че програмата заема много памет и изисква известна практика, за да се научите да използвате всички полезни функции. След като се научите обаче, писането на код на Python с PyCharm ще бъде по-бързо и по-лесно, отколкото с някои други ИСР. Препоръчваме ви да използвате Thonny IDE, ако използвате Python за 3D проектиране, поради възможността да го интегрирате с 3D софтуер (вж. модула *Използване на Python за процедурно генериране на 3D съдържание за 3D печат*). По-долу представяме стъпка по стъпка инсталацията на PyCharm Community в Windows.

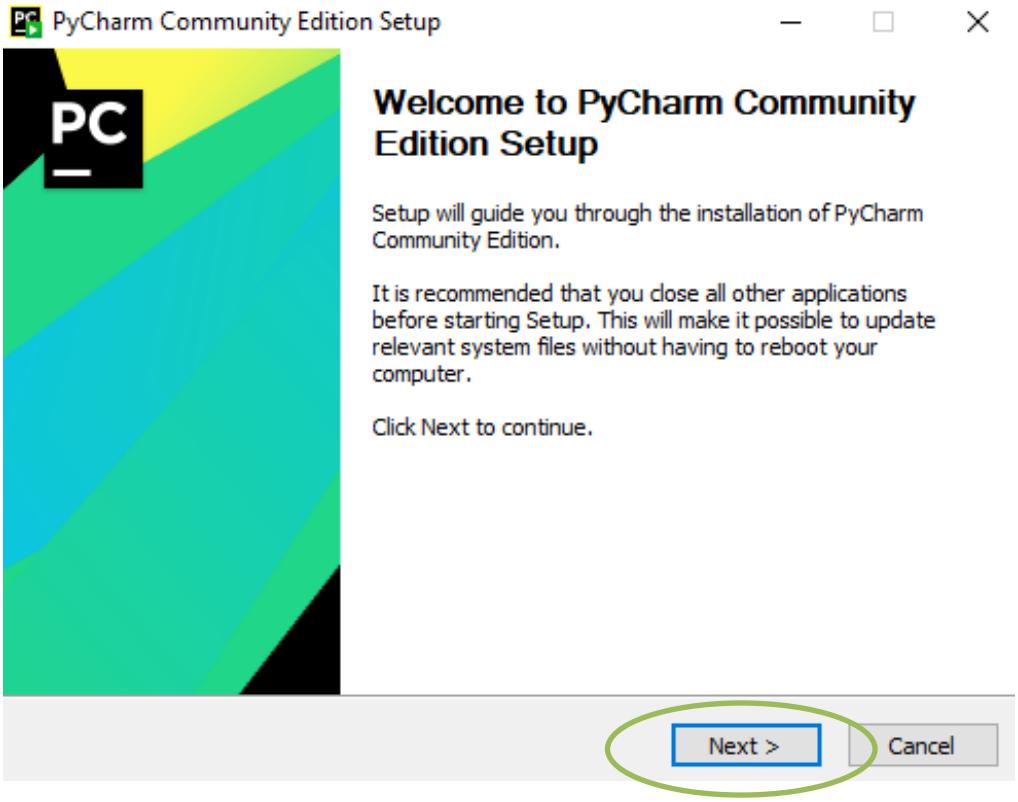
Изтеглете инсталационния файл на PyCharm Community от https://www.jetbrains.com/pycharm/download/#section=windows.



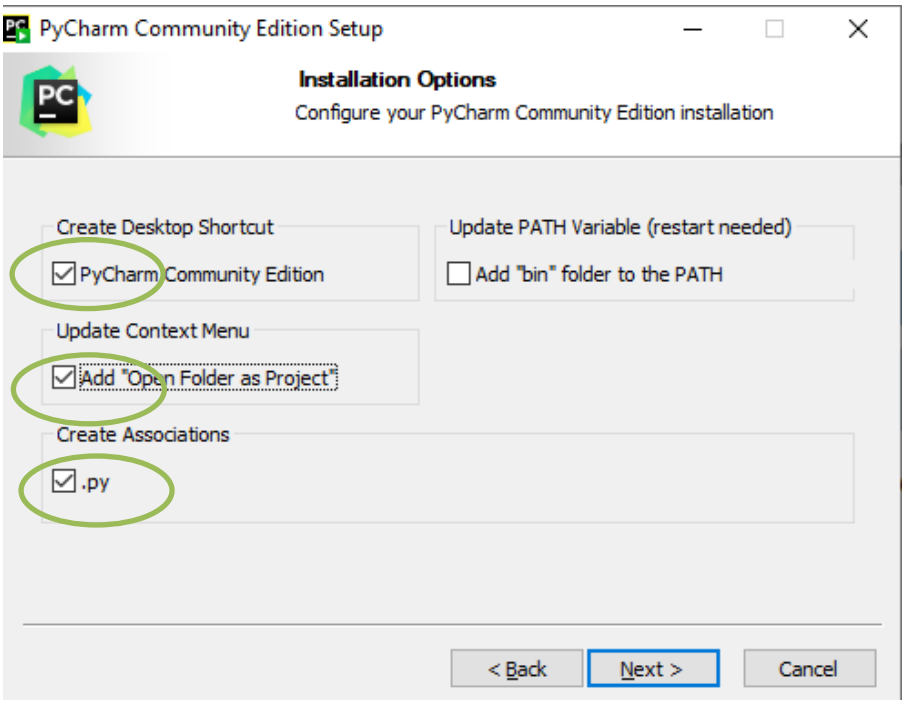
След като стартирате инсталацията, ще се отвори следният прозорец.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



След като кликнете два пъти върху "Next", ще се появи следният прозорец за инсталиране. Изберете опциите, както е показано на снимката по-долу:

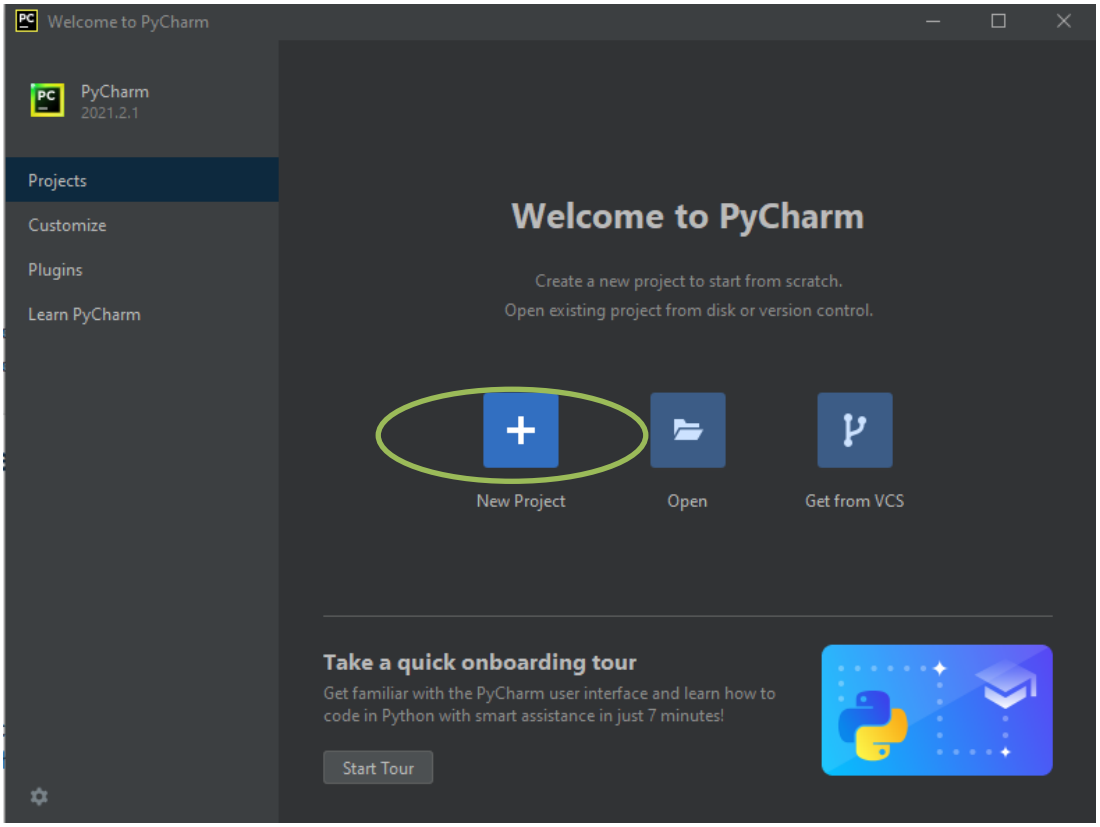


Завършете инсталацията, както е предложено от Инсталатора. Вече сте готови да започнете да използвате PyCharm. Можете да избирате между тъмен и светъл режим и различни цветови схеми по всяко време, според личните си предпочитания. За целта натиснете Ctrl+Alt+S, което отваря настройките, и изберете Editor | Color Scheme. Винаги, когато ви е трудно да си спомните как да използвате някои функции в PyCharm, потърсете в интернет. PyCharm има много полезни функции, достъпни чрез клавишни комбинации.

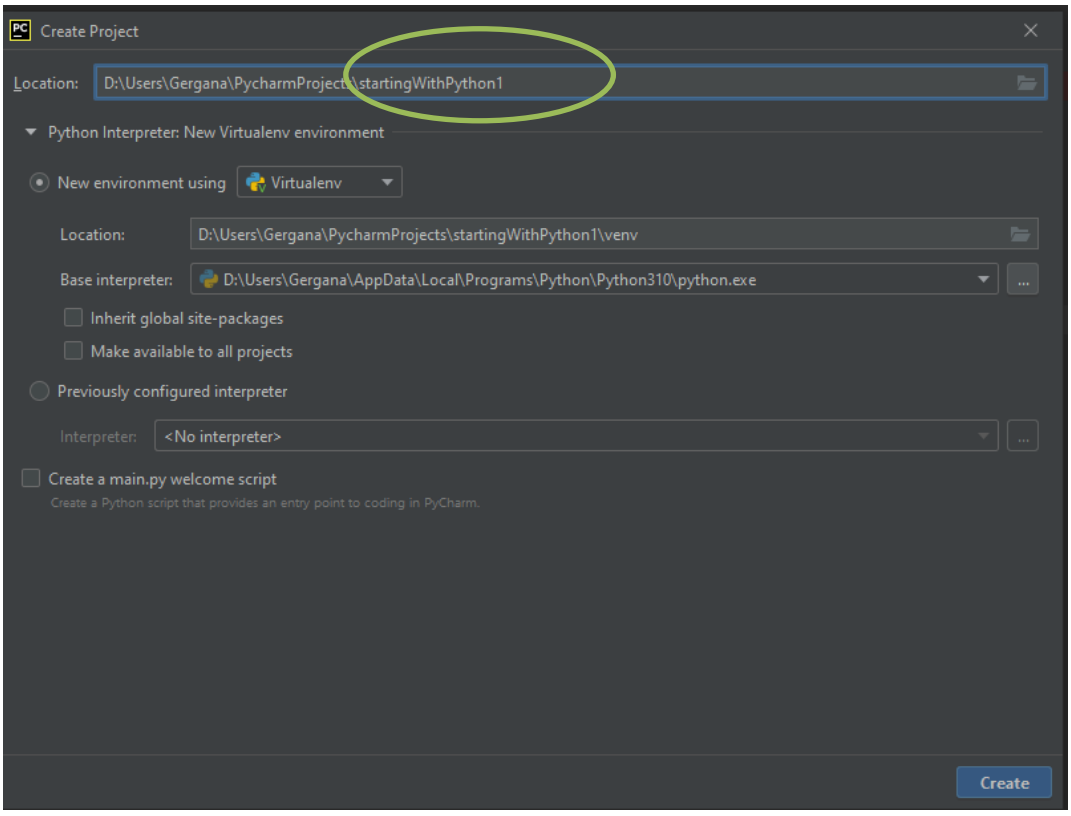
Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

Създайте проект в PyCharm



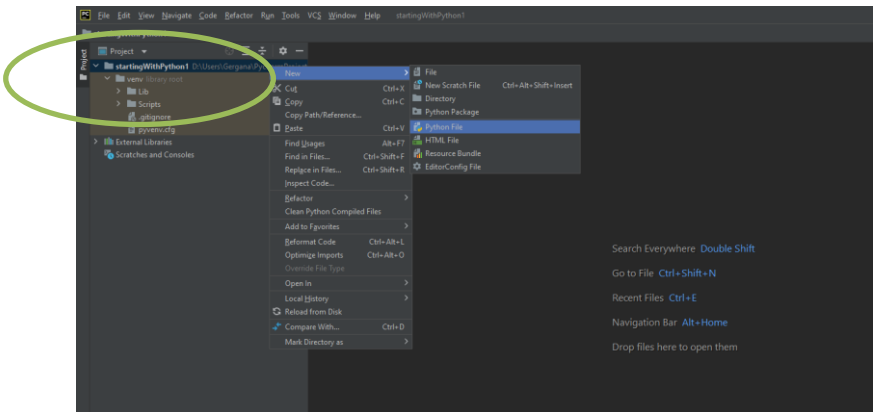
В прозореца, който се отваря, напишете името на вашия проект, оставете всичко останало, както е предложено (въпреки че може да предпочетете да премахнете създаването на main.py), след което кликнете върху "Create".



Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

След като проектът е създаден, създайте нов Python файл, като кликнете с десния бутон на мишката върху името на проекта -> New -> Python File. Дайте име на файла, например example.py.



Регистрирайте се в HackerRank, за да се упражнявате

За да можете да решавате задачи и да проверявате дали са решени правилно, регистрирайте се в HackerRank: https://www.hackerrank.com/dashboard.

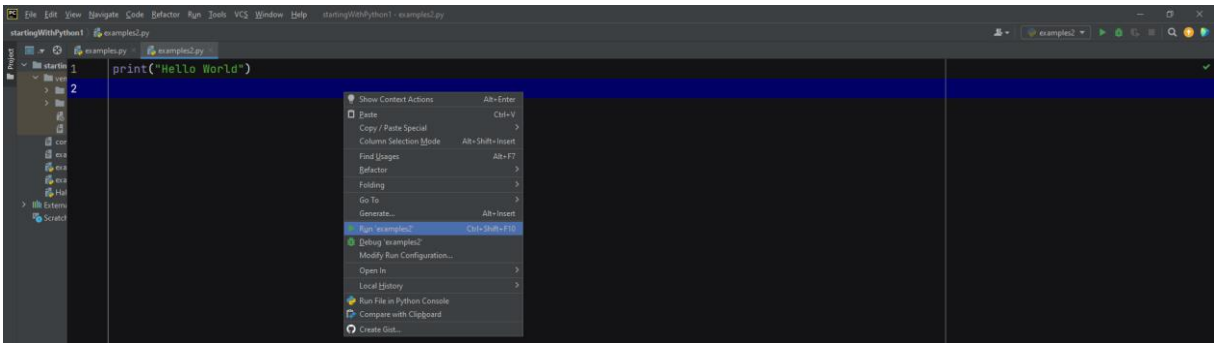
В раздела "*Prepare*" ще намерите много задачи (предизвикателства - challenges), като се започне от най-простите и се стигне до много по-сложни. Запознаването с HackerRank може дори да ви помогне по-късно при интервюта за работа, тъй като понякога на кандидатите се дават задачи за решаване в HackerRank.

Създайте първата си програма

В създадения от вас Python файл въведете следната команда

```
print("Hello World!")
```

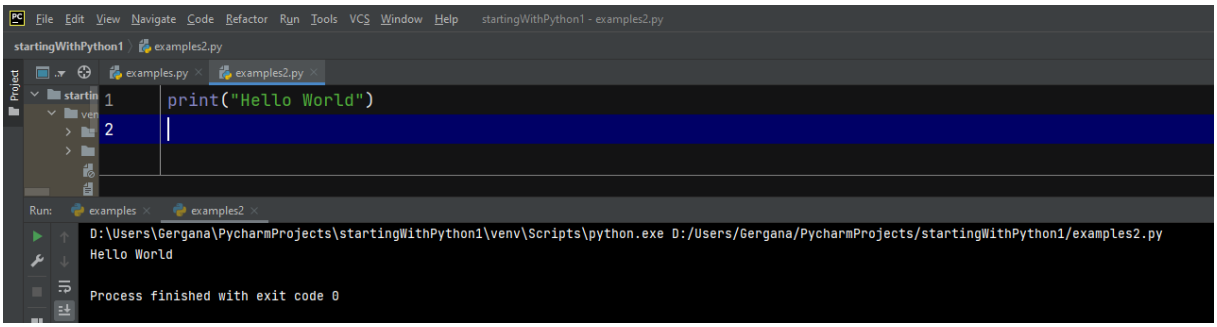
За да стартирате програмата, щракнете с десния бутон на мишката където и да е в прозореца на файла и щракнете върху "Run". Можете също така да използвате клавишната комбинация Ctrl + Shift + F10.



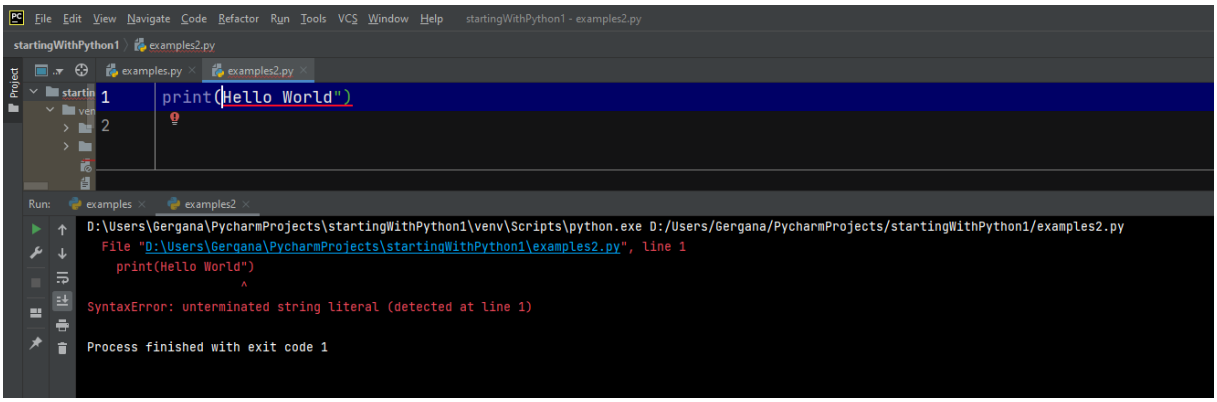
Както виждате, първата ви програма изписва *Hello, World!* на конзолата в долната част на работния прозорец.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

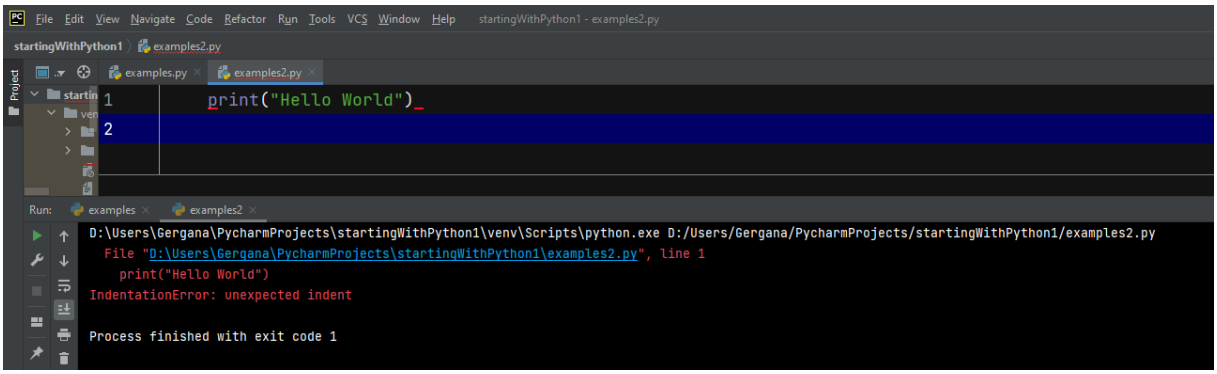
Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Ако сте допуснали някаква грешка в синтаксиса, например забравили сте отварящата или затварящата кавичка, конзолата ще ви съобщи за това и кодът за изход ще бъде 1, вместо 0.



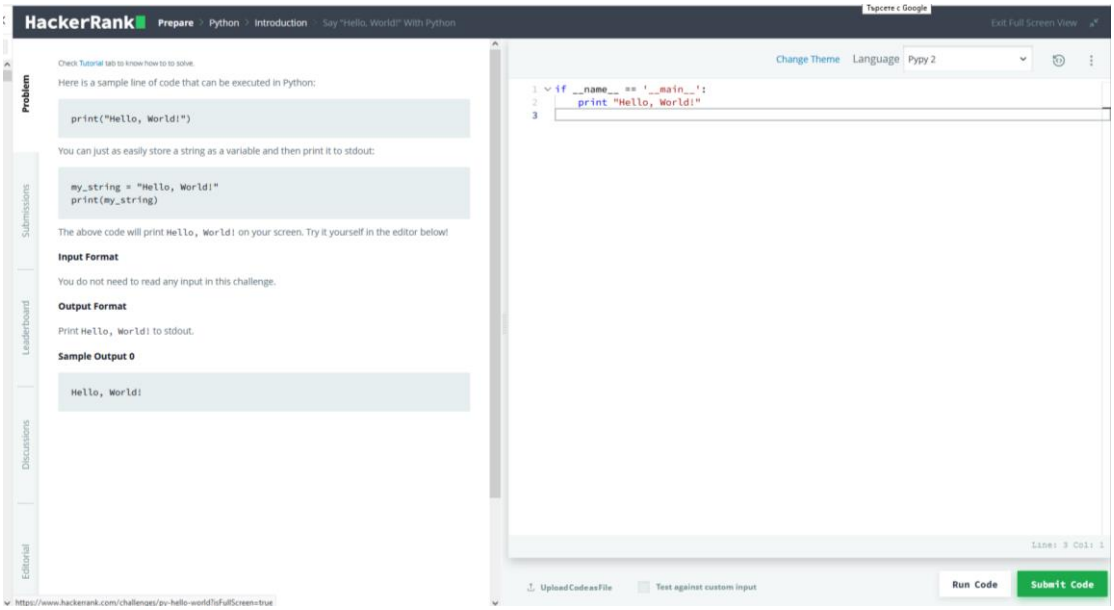
Ако грешката не е в синтаксиса, а в отстъпа преди командата, конзолата също ще ви уведоми, като покаже грешка при отстъп.



Сега можете да се опитате да тествате програмата си и в HackerRank. Намерете предизвикателството “Hello, World!” и въведете кода си.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Променливи и типове данни

Подобно на други езици за програмиране, програмирането в Python разчита на променливи, които са контейнери за съхранение на стойности на данни. Променливите имат име, тип и стойност. В Python има седем вградени типа данни:

- Двоични типове: memoryview, bytearray, bytes
- Тип Boolean: bool
- Типове сет: frozenset, set
- Тип речник: dict
- Типове последователности: range, tuple, list
- Числени типове: complex, float, int
- Тип низови данни: str

В Python няма команда за деклариране на променлива. Променливата се декларира като й се присвоява стойност. Присвояването се извършва със знак =.

```
var = 5
```

На променливата с име var е присвоена стойност 5 и следователно тя е от числов тип.

Възможно е верижно присвояване, при което една и съща стойност може да бъде присвоена на няколко променливи.

```
x = y = z = 15
```

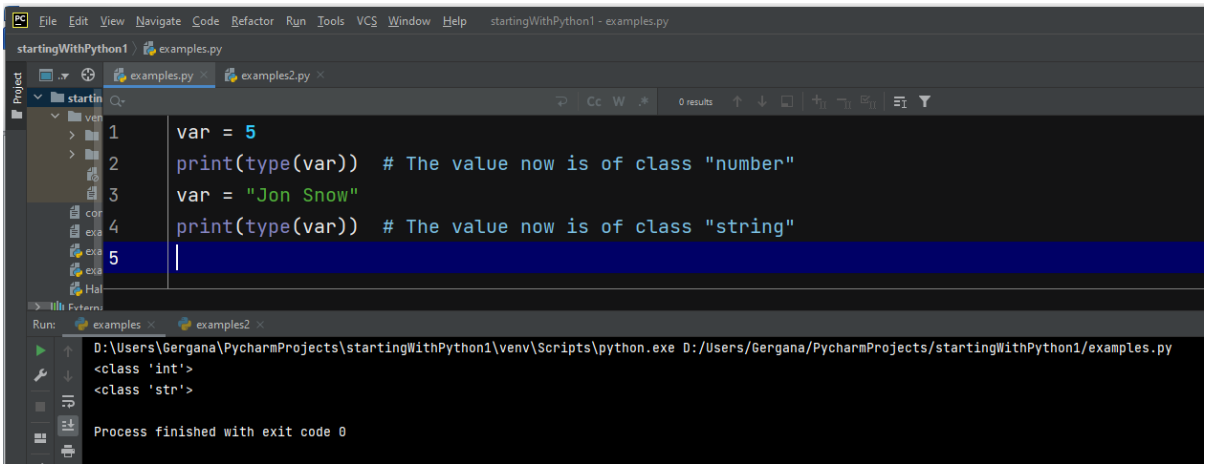
Сега и трите променливи x, y и z имат стойност 15.

В някои езици, като например Java, променливата се декларира като променлива от определен тип (например низ) и въпреки че можем да й присвояваме различни стойности по време на нейния живот в програмата, тези стойности трябва да бъдат от този определен тип данни. Python е динамичен език и това не е необходимо. Типът на данните не е изрично деклариран и зависи само от стойността, която в момента присвояваме на променливата. Ако на вече декларирана променлива се присвои стойност от друг тип,

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

типът на данните ще се промени. В примера по-долу обърнете внимание как типът на класа, който се изписва на конзолата, се променя когато присвояваме нова стойност.



Основни познания

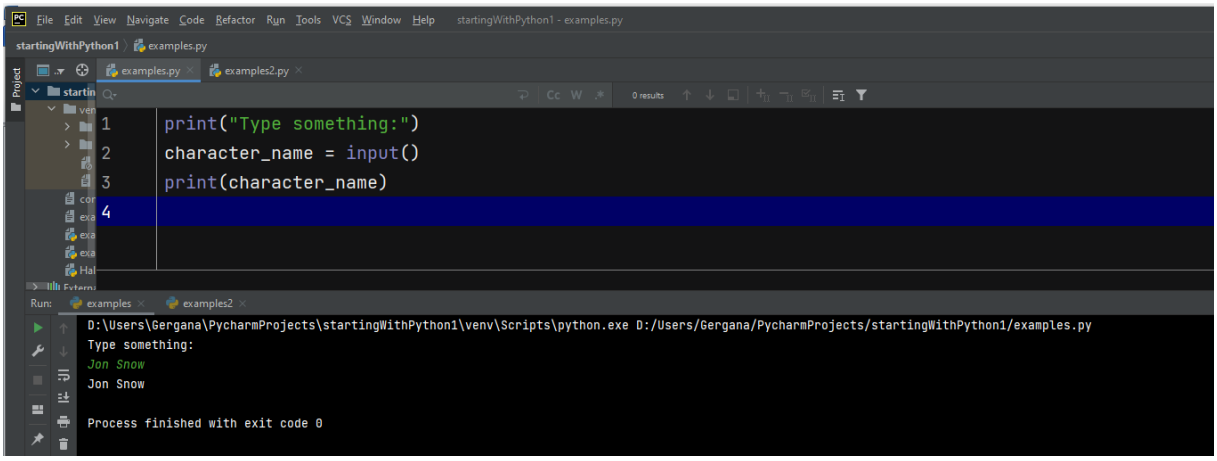
Работа с конзолата

Конзолата работи само с текст. Всичко, което отпечатваме на конзолата, се преобразува в текст. В горния пример видяхте как печатаме на конзолата, а именно с командата print(*име_променлива*). Опитайте пак с кода по-долу и наблюдавайте резултата на конзолата.

```
var = 5

print(var)
```

Можем също така да прочетем всичко, което пишем в конзолата, и да го използваме в нашата програма. Това става с командата input(). Можем да присвоим стойността на input() на променлива и да я използваме в програмата.

Кодът, показан в примера по-долу, ще прочете написаното от потребителя в конзолата и ще го отпечата обратно на конзолата (въведеният от нас текст е в зелено, а отпечатаният текст е в стандартно сиво).

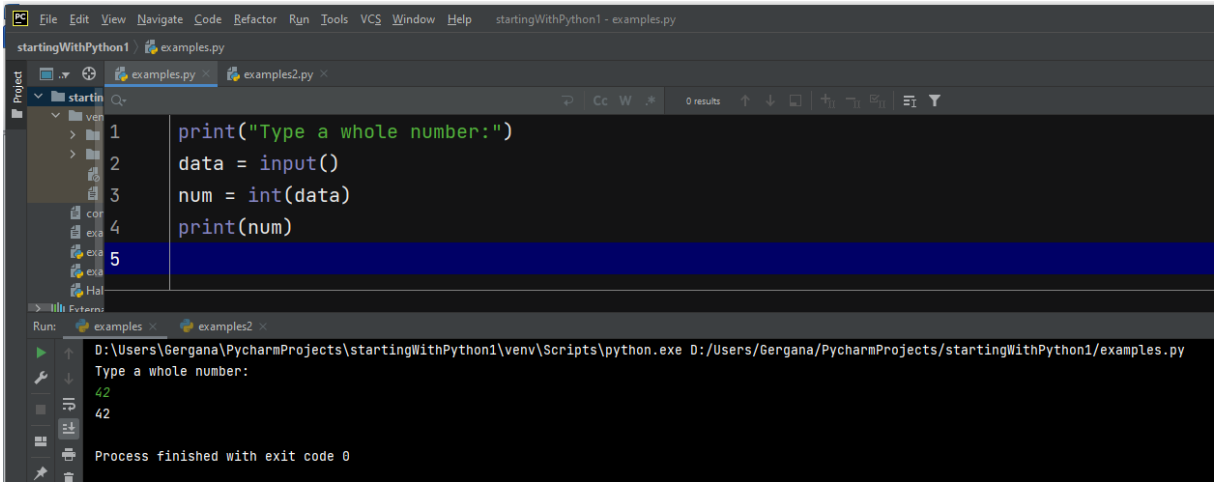


В известен смисъл това беше лесно, защото въведохме текст (“Jon Snow”) и Python автоматично го присвои на променлива от типа *str,* тъй като това е конзолен вход. Но какво ще стане, ако искаме да прочетем число от конзолата и да го присвоим на числова променлива. Тъй като данните, които въвеждаме в конзолата, винаги са текст, трябва да ги

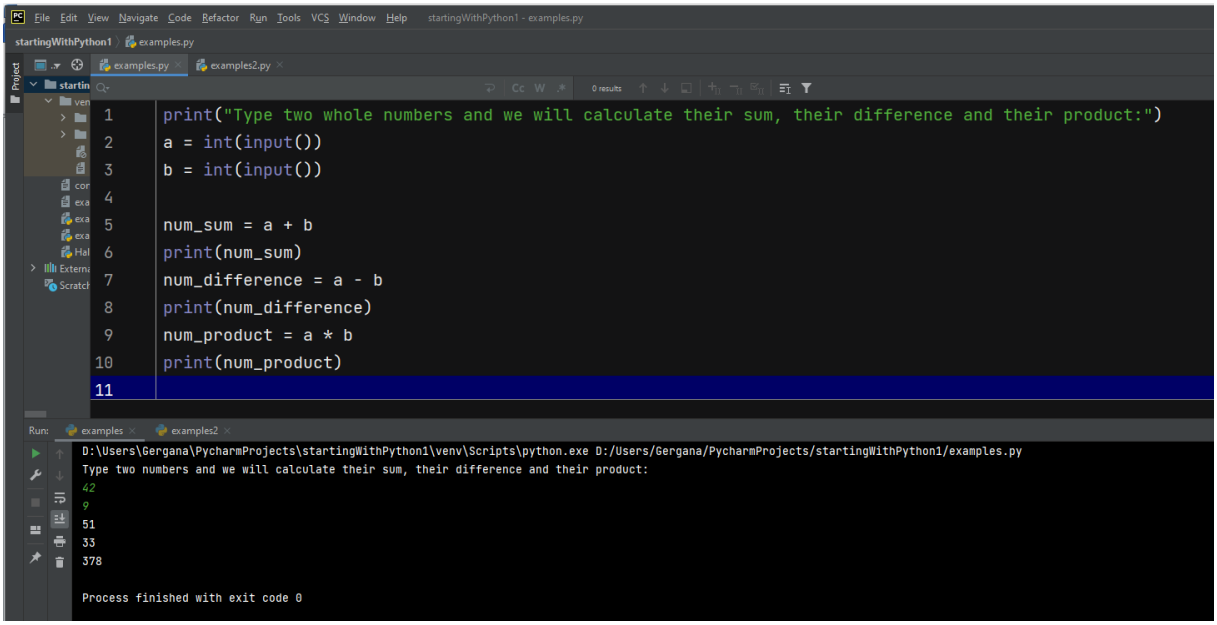
Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

преобразуваме в друг тип данни, ако искаме да присвоим числова стойност на променливата. Преди да превърнем текст в друг тип данни, трябва да сме сигурни, че превръщането ще бъде възможно. Не можем да превърнем текст като "Jon Snow" в числова стойност. В примера по-долу променливата *num* ще бъде числова стойност и ще може да се използва като такава.



Нека видим друг пример по-долу. Тази програма ще прочете две цели числа, които въвеждаме на конзолата, и ще извърши някои основни аритметични операции с тях, а именно събиране (с оператора +), изваждане (с оператора –) и умножение (с оператора *). Резултатите от аритметичните операции ще присвоим на променливи и след това ще можем да ги отпечатаме. По-долу е показан резултатът, който ще получим. Не забравяйте, че входните данни, които въведохме, са показани в зелено, а резултатът е показан в сиво.



Съвет за полезна функционалност на PyCharm:

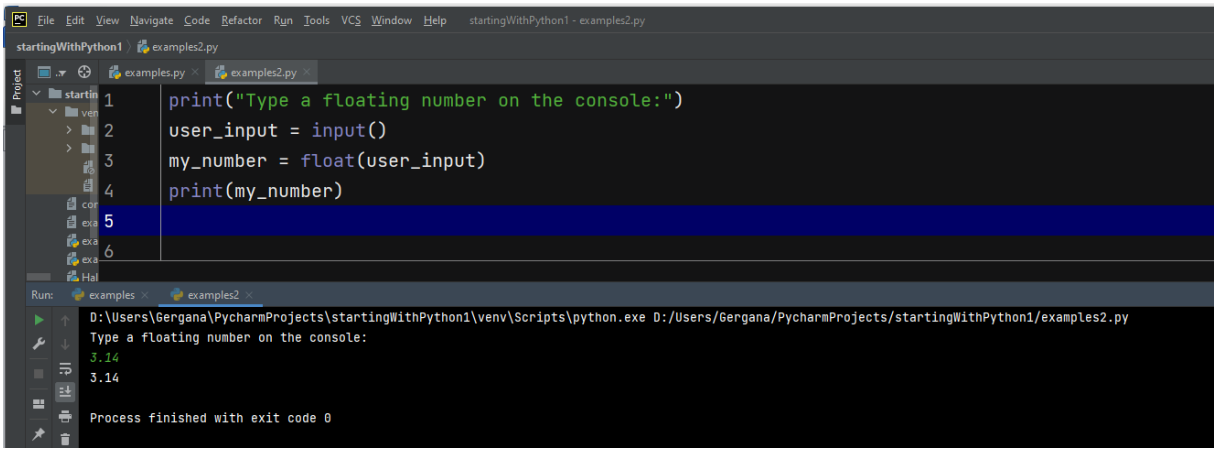
В по-дълги програми в PyCharm можете да използвате комбинацията от клавиши Ctrl+Alt+L, за да форматирате бързо целия си код автоматично и да сведете до минимум грешките при отстъпите.

Възможно е също така да искаме да прочетем десетични дроби (числа с плаващ десетичен знак) от конзолата. Можем да направим това, като превърнем входните данни във *float*. По-

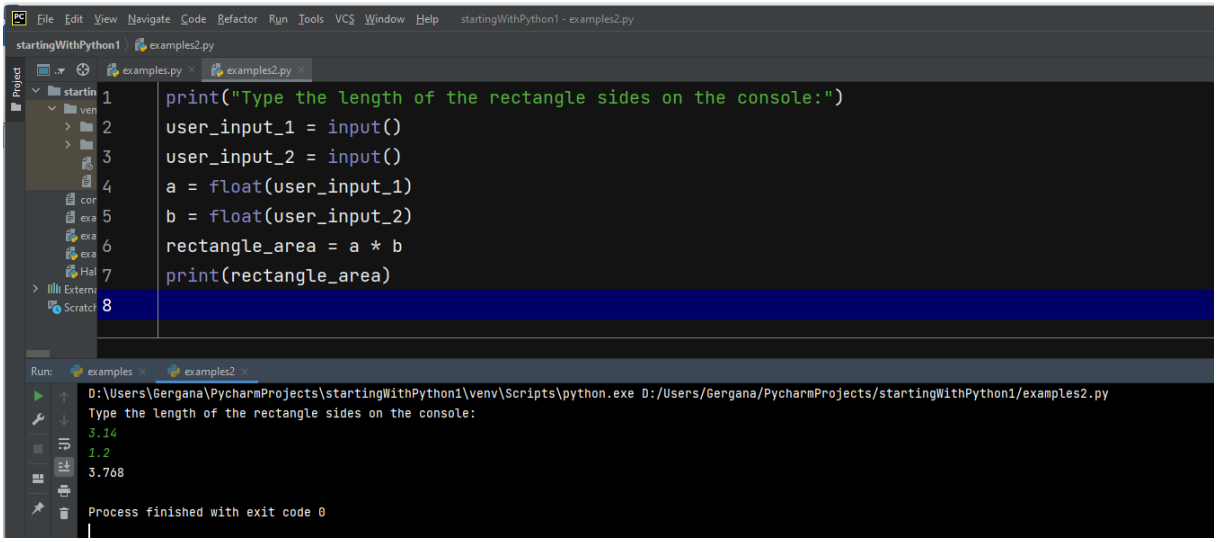
Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

долу ще покажем как да прочетем числото 3.14 от конзолата и да го отпечатаме обратно на конзолата.



В следващия пример се четат две дробни числа, въведени от потребителя, които представляват дължината на двете страни на правоъгълник. След това изчисляваме лицето на правоъгълника (S=a*b). За да прочетем две десетични дроби, въведени на конзолата една след друга, трябва да създадем две променливи с функцията input(). Програмата ще изчака потребителят да въведе толкова числа, колкото сме планирали да прочете, и едва след това ще пристъпи към изчисленията. В нашия пример лицето на правоъгълник със страни 3,14 и 1,2 е 3,768.



Съвет за имената на променливите в Python:

По традиция имената на променливите в Python са малка буква, дума, започваща с малка буква или комбинация от думи, започващи с малки букви. Също така по традиция, след като започнете работа в софтуерната индустрия, от вас ще се очаква да назовавате променливите си на английски език, за да улесните работата в многонационални екипи. Ако се използва повече от една дума, отделяме думите с долна черта, за да подобрим четливостта. Това се нарича стил на писане тип "змия". Използването на повече от една дума в името на променливата е обичайна практика, когато искате да опишете по-добре какво представлява променливата. Това ще помогне на следващия програмист да разбере кода ви по-добре. Например, ще ви позволи да разграничите rectangle_side и triangle_side в една по-сложна програма.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

Стилът на тип “змия”се различава от другите стилове, използвани в други езици за програмиране. Например в Java (който може би познавате от часовете си по информатика) е прието да се използва стил тип “камила”, което би означавало rectangleSide и triangleSide.

Спазването на конвенцията за именуване не е задължително за програмната среда на Python, така че програмата ви ще бъде изпълнена независимо от това как ще наречете променливите си. Следването на конвенциите за именуване обаче ще се очаква, когато започнете работа в индустрията или ако споделяте кода си с други програмисти, така че е препоръчително да свикнете с тях още докато изучавате различни езици за програмиране.

Аритметични операции

Вече използвахме три от четирите основни оператора за аритметични операции в Python, когато се упражнявахме да четем числа от конзолата (вж. по-горе):

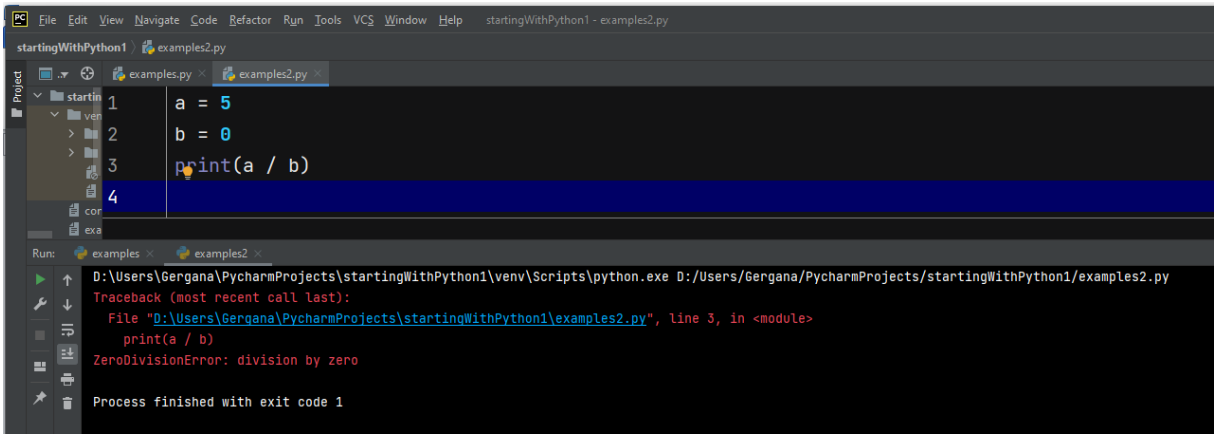
+	Добавяне
–	Изваждане
*	Умножаване

Сега остава само да научим повече за делението. В Python можете да използвате три различни оператора за деление:

/	Обикновено деление, което ще върне точния резултат от делението на две числа, като той винаги ще бъде десетична дроб. Така делението 5/2 ще върне 2,5, а 4/2 - 2,0.
//	Целочислено деление, което ще раздели двете числа, но ще закръгли резултата надолу до най-близкото цяло число. Така делението 5//2 ще върне 2

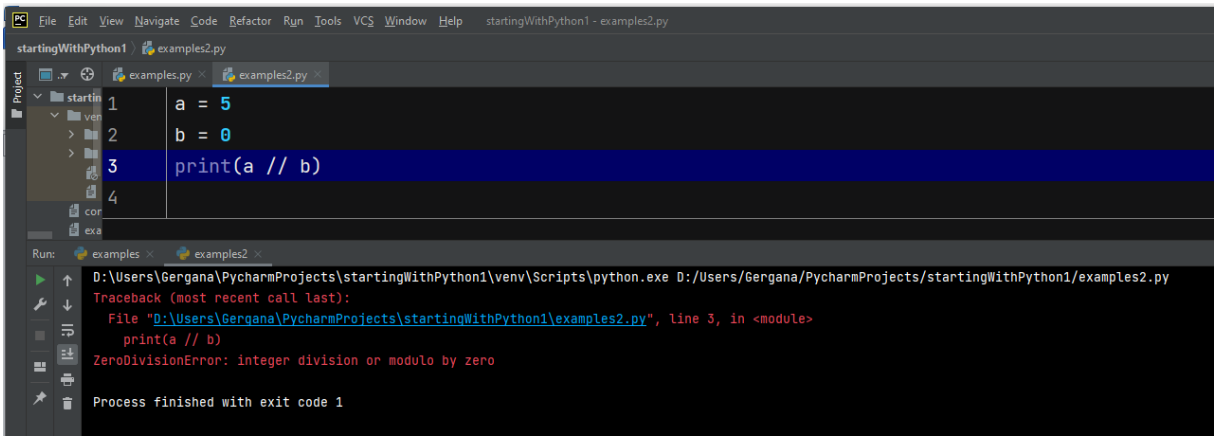
Сега опитайте сами в PyCharm как работят тези два оператора.

Не забравяйте, че деление на 0 не е разрешено. То ще доведе до грешка ZeroDivision и програмата ще се прекрати. Това е важно да се вземе предвид когато създавате вашата програма, ако променливата, която използвате, може потенциално да получи стойност 0. По-долу ще ви покажем примери за деление с 0.

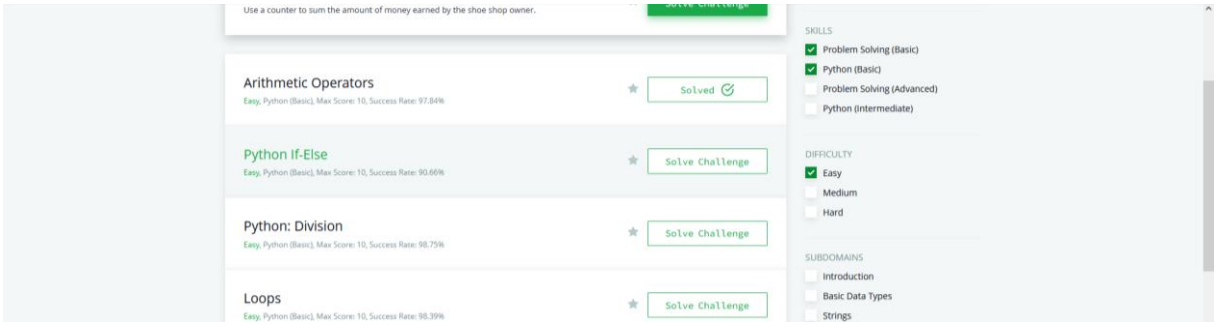


Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



С тези знания сте готови да влезете в HackerRank и да решите предизвикателството Arithmetic Operations и Python: Division (уверете се, че сте избрали ниво на трудност *Easy)*.

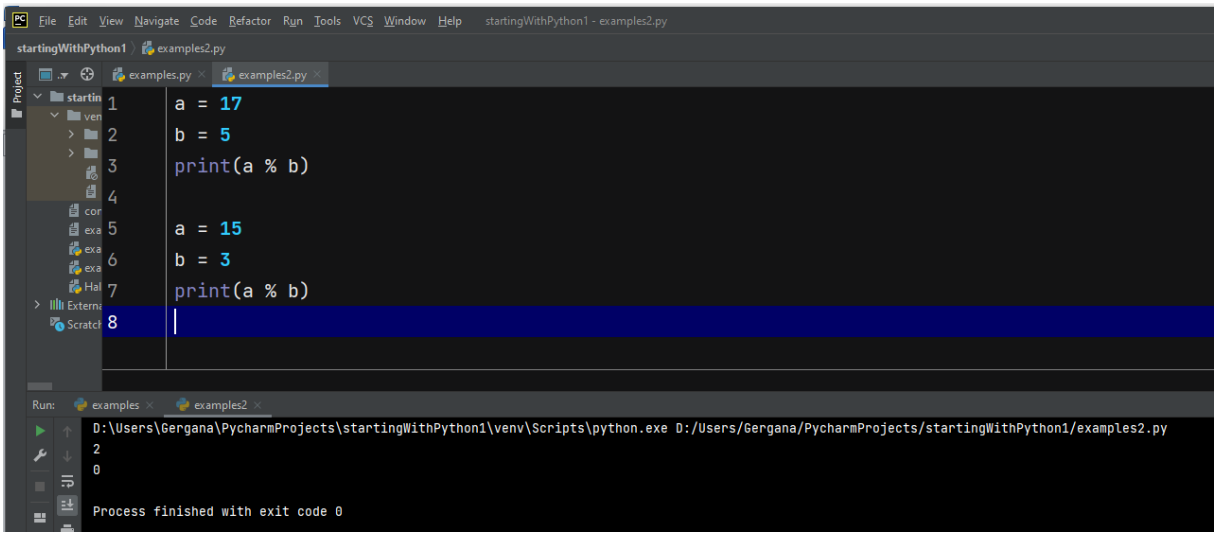


А сега нека се запознаем с последния вид деление, което е възможно в Python.

%	Модулно деление, което ще раздели числата, но няма да върне резултата от делението. То ще върне само остатъка от целочисленото деление. Така делението 5%2 ще върне 1, защото 2 се съдържа в 5 два пъти (2*2=4), но има остатък от 1 (4+1=5). От друга страна, 4%2 ще върне 0, защото 2 се съдържа в 4 два пъти (2*2=4) и няма остатък. Операторът % може да бъде много удобен в комбинация с целочислено деление в случаите, когато трябва да разделим числата на стотици, десетици и единици или когато трябва да разберем дали числото е четно или нечетно.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

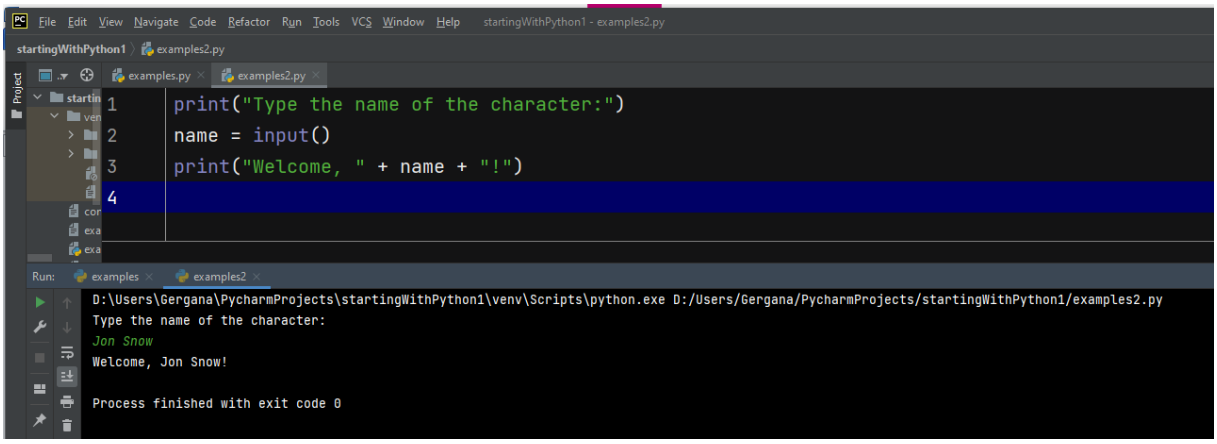
Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Както при целочисленото и обикновеното деление, не можем да извършим модулно деление на 0, защото ще получим грешка ZeroDivision, която ще прекрати програмата.

Отпечатване в конзолата

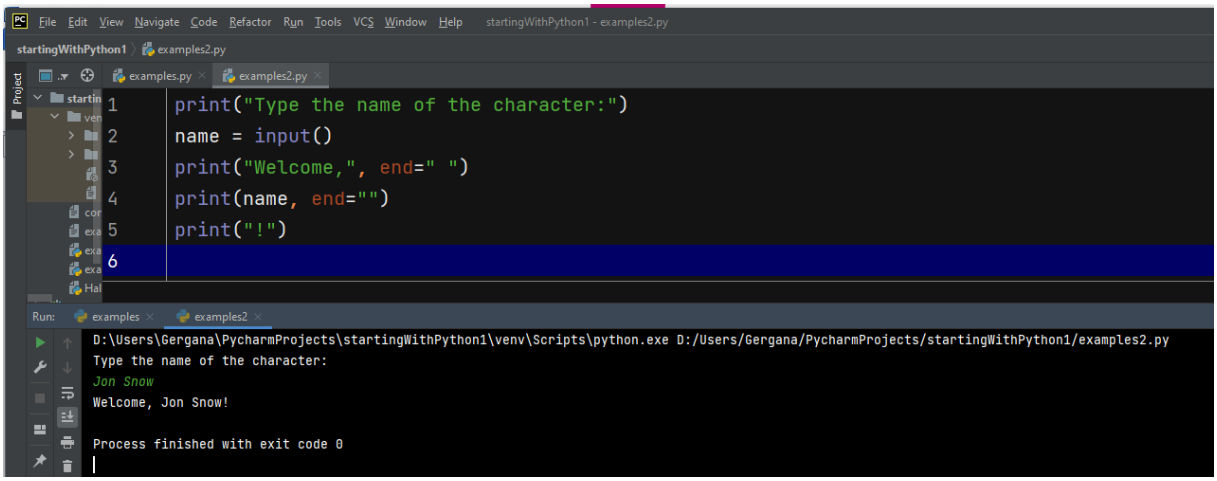
Можем да използваме променливите, прочетени от конзолата, за да отпечатваме по-сложни текстове. Например следният код ще отпечата "Welcome, [*име, предоставено от потребителя*]!". Използваме оператора +, за да комбинираме различните части на текста с входната променлива. Не забравяйте да добавите интервал след "Welcome,", защото в противен случай ще отпечатате "Welcome,[*име, предоставено от потребителя*]!".



По подразбиране командата print() печата нов ред след принтирания текст. Можете обаче да промените това, като посочите, че краят на отпечатването трябва да бъде например празен интервал или без интервал. Можете да направите това, като присвоите желания край на печата на променливата end и я въведете като допълнителен аргумент във функцията print(). В примера по-долу ще отпечатаме "Welcome, [*име, предоставено от потребителя*]!", като използваме този подход. Недостатъкът на този подход е, че извикваме функцията print() 3 пъти.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

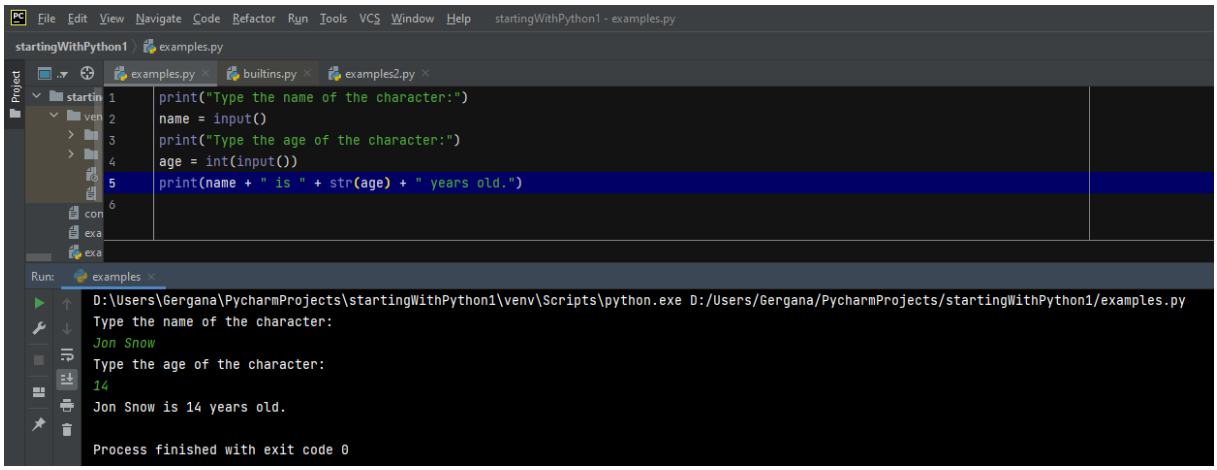


Съвет за полезна функционалност на PyCharm:

В PyCharm можете бързо да дублирате един ред, като поставите курсора си някъде върху реда, който искате да дублирате, и натиснете клавишната комбинация Ctrl+D. Кодът ще бъде дублиран точно под избрания ред. Ако селектирате част от реда и след това натиснете Ctrl+D, ще дублирате селекцията. Това помага за по-бързото писане на код, ако кодът се повтаря.

Малко по-сложно е, когато искаме да отпечатаме променливи, които са числа. В този случай, за да преобразуваме числото в текст, използваме функцията str() и въвеждаме името на числовата променлива като аргумент (като го поставим в скобите на функцията). Тя работи както за цели, така и за дробни числа, стига въведените от потребителя данни да са преобразувани в десетична дроб чрез float(input()).

```
age = int(input())
str(age)
```



Съвет за опростяване на кода:

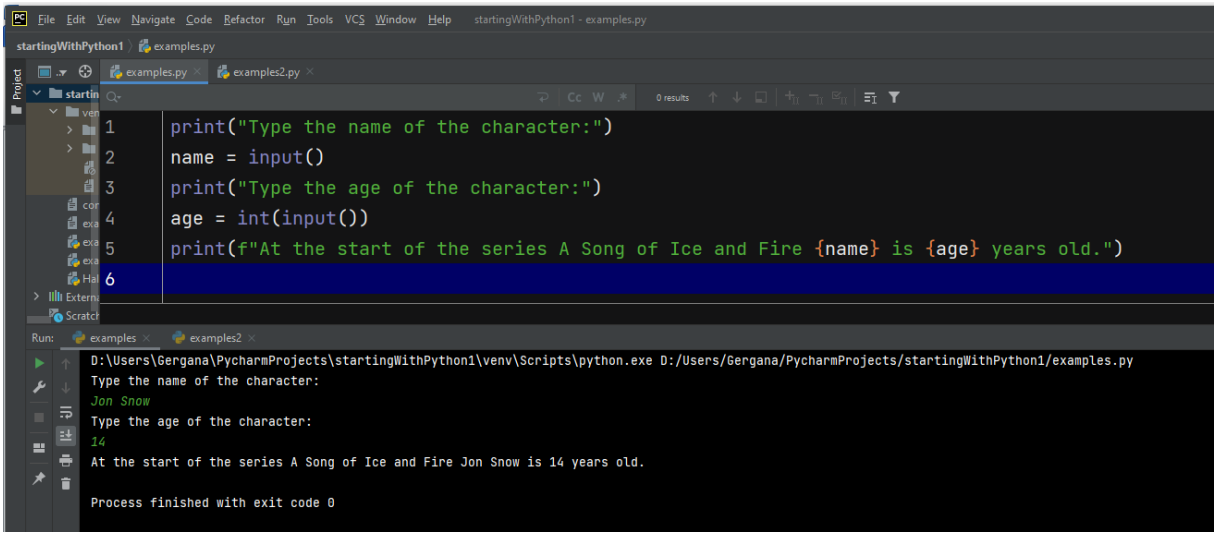
Не е необходимо първо да присвояваме резултата от функцията input() на променлива и след това да подаваме променливата като аргумент във функцията print(). Можем просто директно да подадем input() като аргумент във функцията print(): т.е. print(input())

Използването на оператора + за отпечатване е интуитивно, но кодът става доста нечетлив. Макар че в този малък пример това няма значение, представете си, че трябва да комбинирате текста от 10 променливи... За щастие има и по-опростен начин, който се

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

основава на форматиране на изходния низ (известен още като *интерполация на низове*). От Python 3.6 нататък интерполацията на низове може да се извършва чрез поставяне на символа f пред текстовия низ и поставяне на текста и всички "контейнери" на променливи между кавичките на низа. "Контейнерът" на променлива се състои от името на променливата, поставено в къдрави скоби, а именно {*име на променлива*}. За да разберете подхода, вижте примера по-долу. Вече не е необходимо да се притесняваме за оператора + или да добавяме празно място. Имената на променливите в къдравите скоби ще бъдат заменени с въведените от потребителя данни от конзолата. Съществуват и други начини за форматиране на текст, които ще разгледаме в раздела "Работа с текст".



Импортиране на библиотеки

Python разполага с много библиотеки, които съдържат голям брой полезни сложни функции, които вече са реализирани и могат просто да бъдат извикани и използвани в нашите програми. За да направим това, трябва да ги импортираме в програмата. Това се прави в началото на Python файла чрез използване на ключовата дума import, последвана от името на библиотеката.

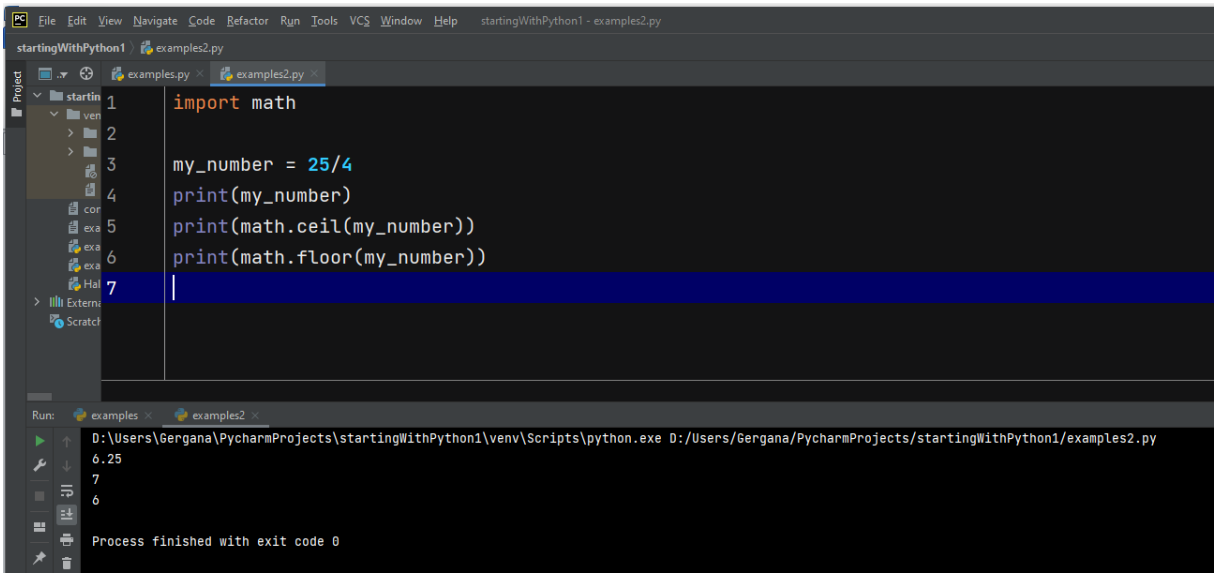
```
import math
```

Това изявление ще зареди всички стандартни математически константи и функции, налични в библиотеката math, и ще можем да ги използваме в нашата програма.

В примера по-долу импортираме библиотеката *math* и извикваме две от нейните функции - ceil() и floor(). Те се извикват, като името им се добавя след math с точка: math.ceil(*нашето число*) закръгля подаденото като аргумент дробно число *нагоре* до най-близкото цяло число, а math.floor(*нашето число*) закръгля дробното число *надолу* до най-близкото цяло число.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

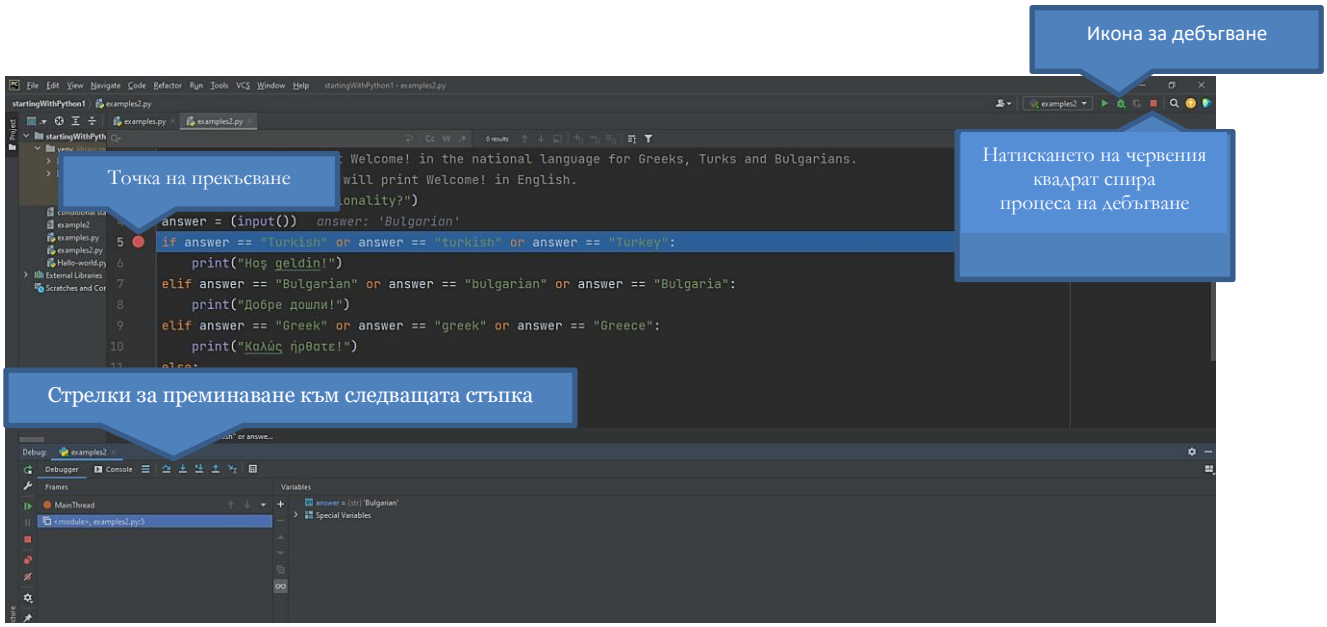


Дебъгване (отстраняване на грешки)

Когато пишем сложни програми, често ни се налага да проследим изпълнението на програмата стъпка по стъпка, за да открием къде нашият алгоритъм греши. Можем да направим това, като рестартираме програмата си не със зелената стрелка, а с иконата, която прилича на буболечка. За да дебъгнем програмата, трябва да посочим точка на прекъсване в програмата, след която PyCharm ще ни покаже изпълнението стъпка по стъпка. Ако не посочим точка на прекъсване, програмата ще се изпълни както обикновено. Обикновено посочваме точката на прекъсване точно преди мястото в програмата, където очакваме да се получи грешка. Точката на прекъсване се поставя като просто кликнем върху сивото пространство в лявата част на програмата, както е показано на скрийншота по-долу. След като започнете процеса на дебъгване, в конзолата ще бъде показана информация, която ще ви помогне да разберете какво се случва с различните променливи и да определите къде алгоритъмът е грешен. Можете да се придвижвате напред и назад в последователните стъпки със стрелките над тази информация. За да прегледате стъпка по стъпка изпълнението на отделен блок от кода, трябва да изберете “Step into” (това е много полезно, когато дебъгвате цикли, които ще разгледаме скоро). За да преминете към следващия блок, трябва да изберете “Step over”. Обикновено отнема известно време да се научите да дебъгвате ефективно, но с постоянни упражнения ще започнете да се справяте по-добре.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Работа с числа

Закръгляне на числата

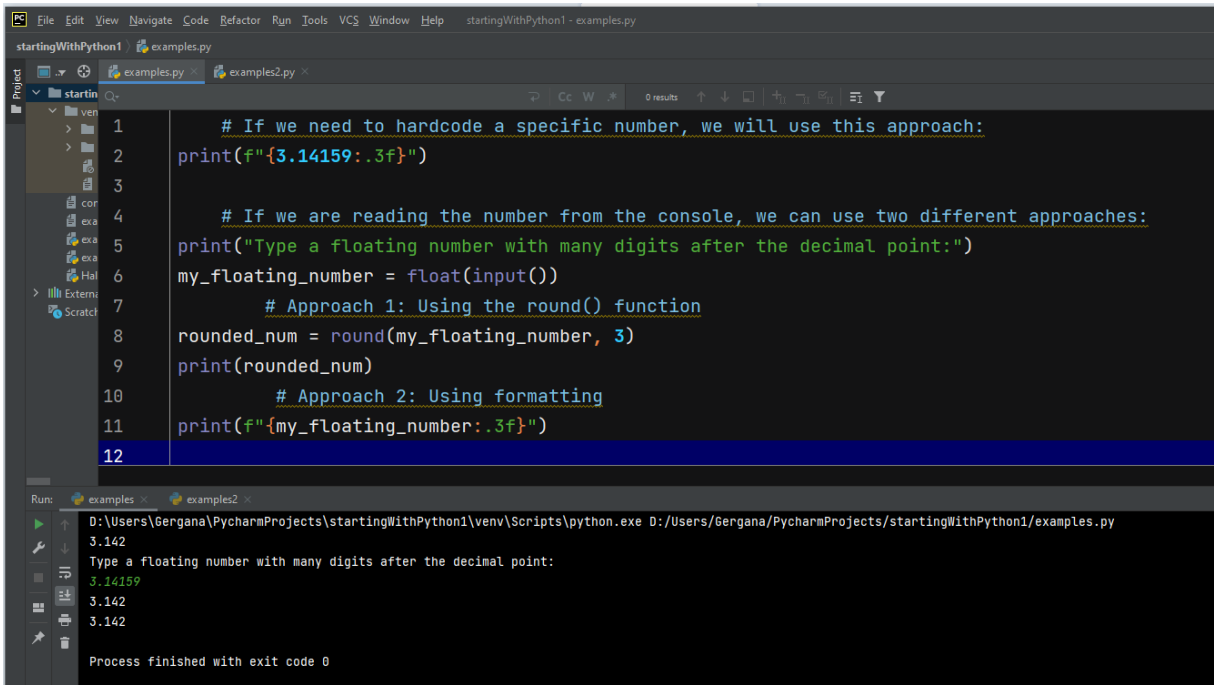
Често в програмите се налага да закръгляме числа. Вече научихме как можем да закръгляме числата нагоре или надолу до най-близкото цяло число. По-често обаче ни се налага да върнем на програмата число, закръглено до определен брой цифри след десетичната запетая. В Python това може да се постигне по два начина, като изборът зависи от програмата и от това какво ще правим с изхода. Първият вариант е да се използва функцията round(*моето число, брой цифри след десетичния знак*), в която трябва да подадем като аргументи дробното число и броя цифри след десетичния знак, които искаме да покажем. Вторият вариант е да се използва допълнително форматиране при интерполацията на низа. Това е показано по-долу:

```
f"{my_floating_number: .2f}"
```

В горния пример можем да заменим "2" с произволно число, в зависимост от това колко цифри след десетичния знак искаме да покажем. По-долу можете да видите пример за форматиране на директно зададено число и пример за форматиране на число, зададено чрез променлива. И двата подхода извеждат закръгленото число *пи*.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Абсолютна стойност

Понякога в програмите се налага да използваме абсолютната стойност на числото - стойността без значение на знака (т.е. неотрицателната стойност на числото). Това може да се направи с функцията abs(*моето число*). По този начин abs(-5) ще бъде равно на abs(5). Опитайте сами в PyCharm.

Условни конструкции

Условните конструкции са в основата на алгоритмите за програмиране, тъй като те ни позволяват да вземаме решения за това как ще продължи програмата и какво ще направи тя въз основа на това дали дадено условие е вярно (True) или невярно (False).

Проверка дали дадено условие е True или False

За да решим дали дадено условие е True или False, трябва да извършим някои проверки, които сравняват стойности. Основните оператори за такива сравнения са:

Оператор	Име	Обяснение	Примери *Да приемем, че:* *a=20* *b=40*
==	Равно	Връща True, ако две стойности са равни, и False, ако са различни	print(a==b) #False print(2*a==b) #True
!=	Не е равно	Връща True, ако две стойности са различни, и False, ако са равни	print(a!=b) # True print(a!=2*b) # True print(2*a!=b) #False
>	По-голямо от	Връща True, ако първата стойност е по-голяма от втората стойност	print(a>b) # False

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

			print(b>a) #True print(2*a>b) # False
>=	По-голямо или равно на	Връща True, ако първата стойност е по-голяма или равна на втората стойност	print(a>=b) # False print(b>=a) #True print(2*a>=b) # True
<	По-малко от	Връща True, ако първата стойност е по-малка от втората стойност	print(a<b) # True print(b<a) # False print(2*a<b) # False
<=	По-малко или равно на	Връща True, ако първата стойност е по-малка или равна на втората стойност	print(a<=b) # True print(b<=a) # False print(2*a<=b) # True
в	Елемент, присъстващ в поредица	Изчислява дали посоченият елемент е в дадена последователност	'o' **in** 'Jon' # True

Сравняването на числата е подобно на основните математически операции. Въпреки това тези оператори могат да се използват и за сравняване на нечислени видове стойности, като дати и низове. Сравняването на низове в Python се извършва на базата на символите в двата низа, като символите се сравняват един по един. Когато се открият различни символи, Python сравнява техните ASCII стойности. За да разберете това, трябва да сте наясно, че низът е последователност от символи (знаци). Компютрите обаче не разбират символите - те съхраняват и манипулират всички данни като 0 и 1. Затова един символ се идентифицира с двоично число, както е показано в ASCII таблицата по-долу - за всеки символ имаме конкретно число.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

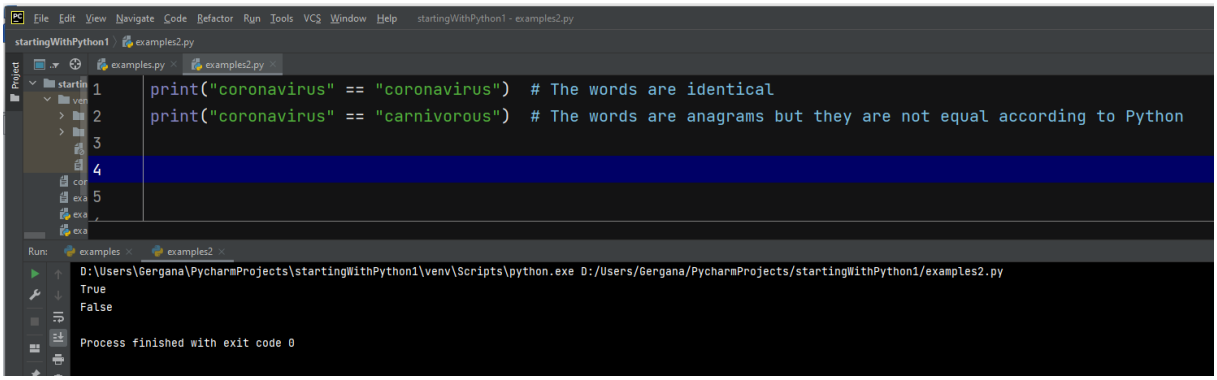
Char	Dec	Binary	Char	Dec	Binary	Char	Dec	Binary
!	033	00100001	A	065	01000001	a	097	01100001
"	034	00100010	B	066	01000010	b	098	01100010
#	035	00100011	C	067	01000011	c	099	01100011
$	036	00100100	D	068	01000100	d	100	01100100
%	037	00100101	E	069	01000101	e	101	01100101
&	038	00100110	F	070	01000110	f	102	01100110
'	039	00100111	G	071	01000111	g	103	01100111
(	040	00101000	H	072	01001000	h	104	01101000
)	041	00101001	I	073	01001001	i	105	01101001
*	042	00101010	J	074	01001010	j	106	01101010
+	043	00101011	K	075	01001011	k	107	01101011
,	044	00101100	L	076	01001100	l	108	01101100
-	045	00101101	M	077	01001101	m	109	01101101
.	046	00101110	N	078	01001110	n	110	01101110
/	047	00101111	O	079	01001111	o	111	01101111
0	048	00110000	P	080	01010000	p	112	01110000
1	049	00110001	Q	081	01010001	q	113	01110001
2	050	00110010	R	082	01010010	r	114	01110010
3	051	00110011	S	083	01010011	s	115	01110011
4	052	00110100	T	084	01010100	t	116	01110100
5	053	00110101	U	085	01010101	u	117	01110101
6	054	00110110	V	086	01010110	v	118	01110110
7	055	00110111	W	087	01010111	w	119	01110111

Източник: https://towardsdatascience.com/processing-text-with-unicode-in-python-eacc226886cb

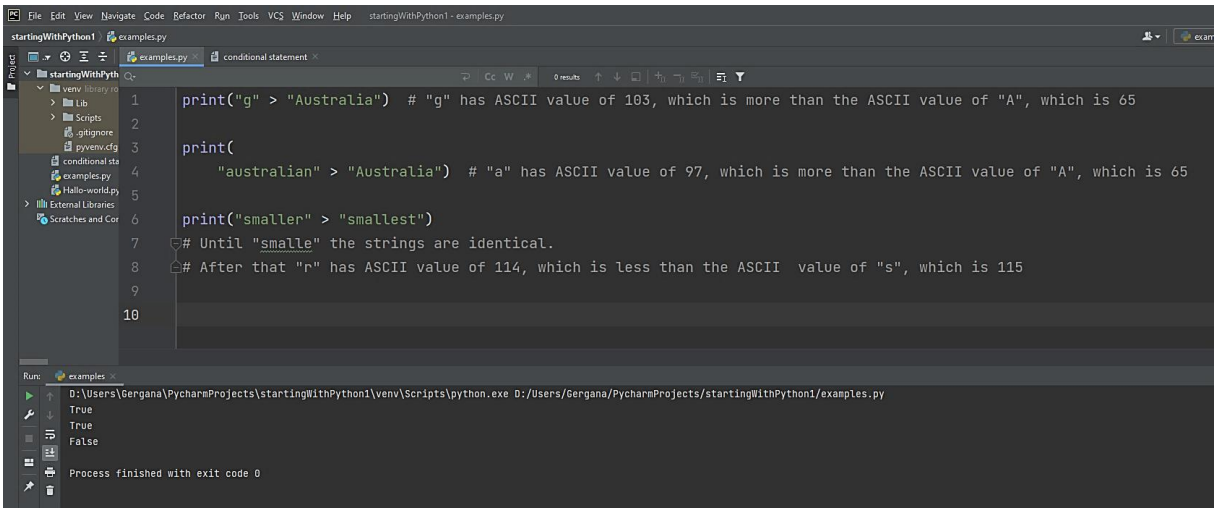
Сравнението с помощта на операторите == и ! = е съвсем интуитивно. Два символни низа ще бъдат равни само ако всички техни символи са идентични и са разположени на еднакво място. Те ще бъдат различни, ако има поне един различен символ или позиционирането на символите е различно. Сравнението е чувствително към големи и малки букви, тъй като големите букви имат различни ASCII стойности от малките. Анаграмите - думи, съдържащи едни и същи символи, но в различен ред, няма да бъдат еднакви, защото символите се сравняват един по един и позиционирането им има значение. Вижте някои примери по-долу.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Когато използвате операторите > , < , >= и <=, символът с по-ниска ASCII стойност се счита за по-малък. Python ще сравни ASCII стойностите на първите различаващи се символи в двата низа и тази стойност ще реши кой низ е по-голям от другия. Стойностите на символите след това не се взимат предвид. Вижте примерите по-долу.

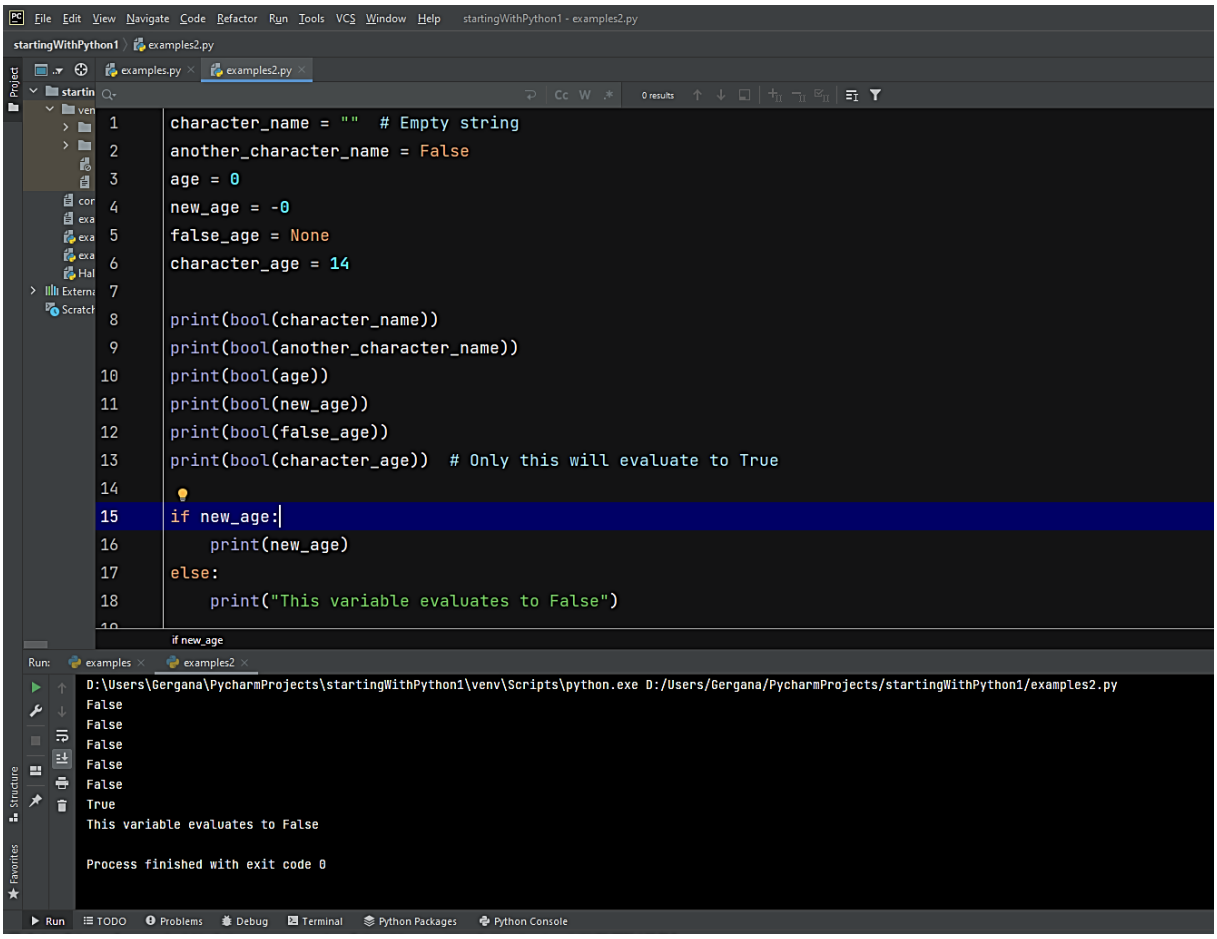


Съвет за True и False стойности на променливи:

Самата променлива също може да се оцени като True или False. Ако променливата има стойност, тя винаги се оценява като True. Ако тя е декларирана като None, 0, -0, “”, ‘’ или False, тогава тя ще се оцени като False. Така можем да използваме променлива, за да получим булева променлива, или можем да я използваме в условни изрази (условните оператори ще разгледаме в следващия раздел, така че ако не разбирате това, върнете се към него. Условните оператори обаче са много сходни във всички езици за програмиране). Вижте по-долу.

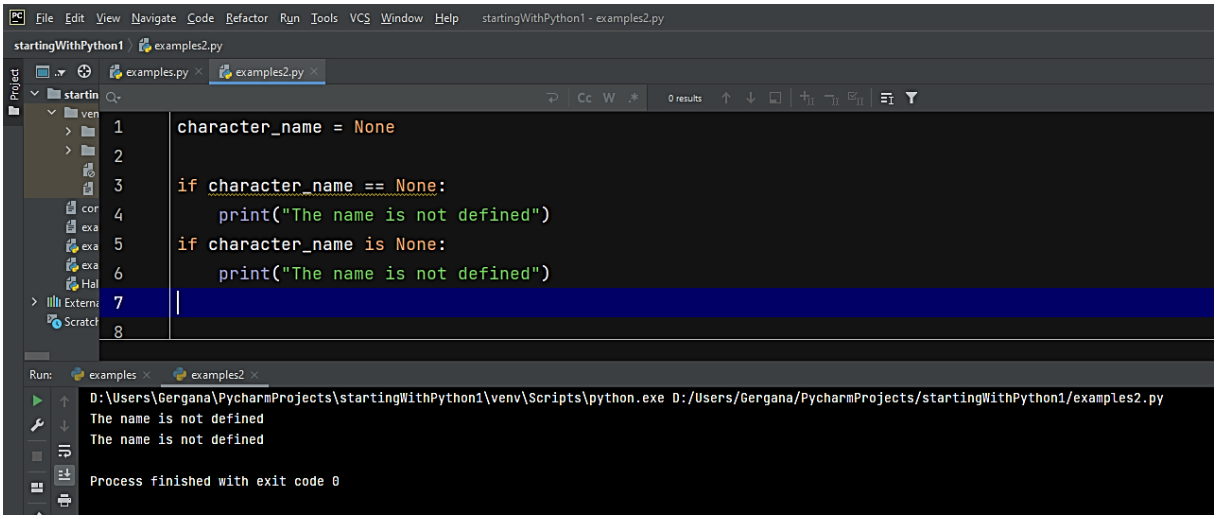
Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Съвет относно ключовата дума None:

None е ключова дума, която дефинира нулева стойност или липса на стойност. Можем да присвоим None на променлива. В условна конструкция също можем да проверим дали дадена променлива е None. За тази цел сравняваме променливата и None или с оператора =, или с ключовата дума is. Предпочитаният синтаксис е с ключовата дума is.



Съфинансиран от програма „Еразъм+“ на Европейския съюз

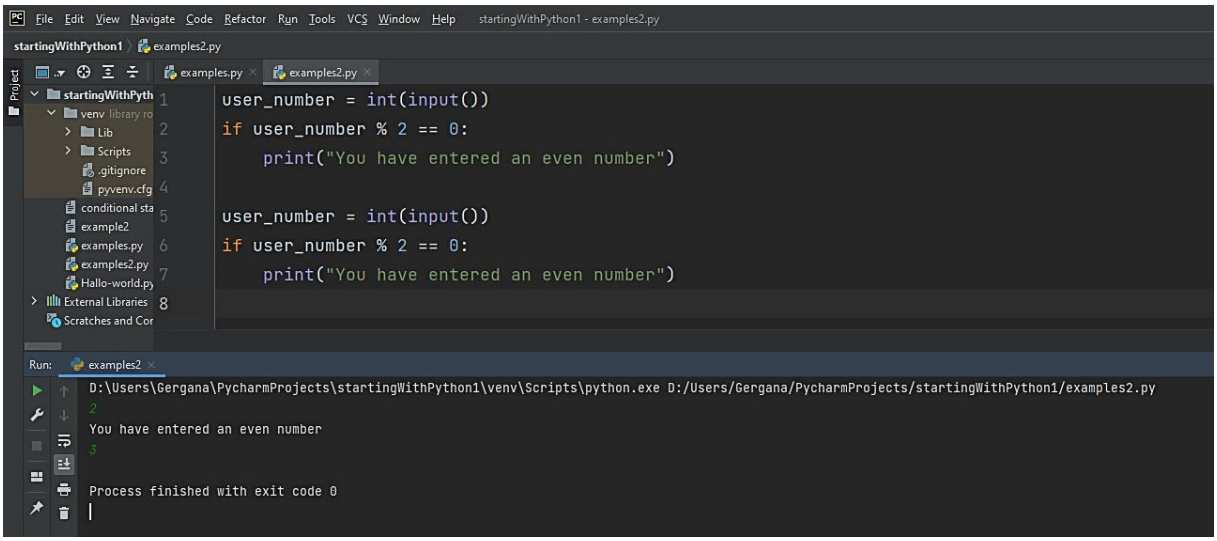
Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

Използване на условни оператори

След като сме в състояние да решим дали дадено условие е True или False, можем да конструираме подробна програмна логика въз основа на прости алгоритми. За тази цел използваме следните условни оператори и конструкции:

- if
- if-else
- if-elif-else
- вложени if-else конструкции

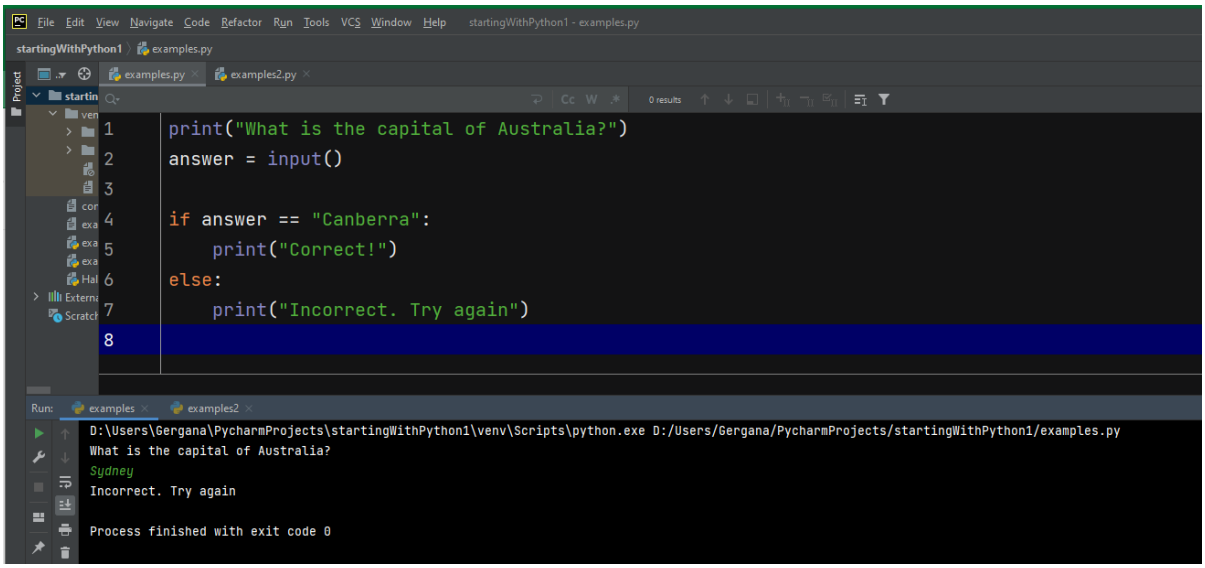
Когато използваме оператора if, проверяваме валидността на условието и въз основа на това решаваме дали даден блок от код ще бъде изпълнен. Да кажем, че искаме потребителят да въведе число и го уведомяваме, ако числото е четно. Можем да направим това, като използваме модулно деление на 2. Четното число няма да върне остатък, т.е. резултатът ще бъде 0. По-долу можете да видите подробното решение. Обърнете внимание, че използваме if и подаваме в него оператор, който проверява дали дадено условие е True или False. След този оператор добавяме двоеточие и на нов ред, с отстъп, записваме блока код, който искаме да се изпълни, ако условието е True. Ако условието е False, кодът, посочен след двоеточието, няма да бъде изпълнен. В примера по-долу на конзолата първо въвеждаме четно число и получаваме съобщението. След това въвеждаме нечетно число и не получаваме съобщение.



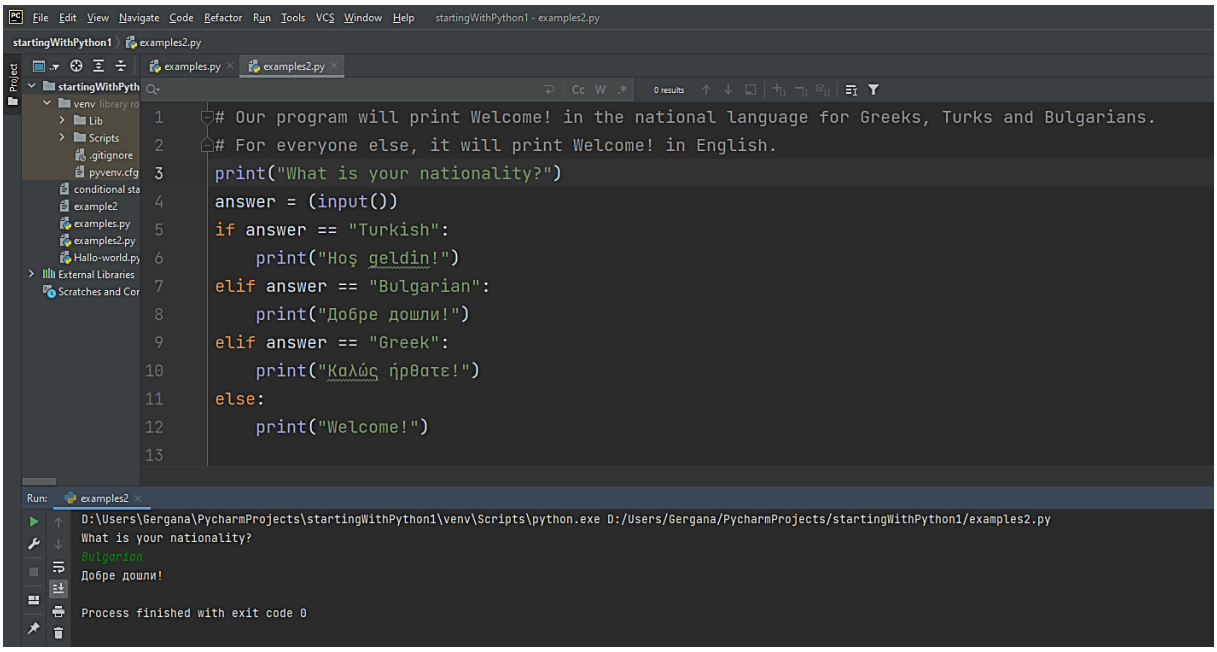
Най-често обаче, когато дадено условие не е Вярно, бихме искали да изпълним различен блок от код. Ако случаят наистина е такъв, трябва да използваме оператора if-else. Блокът код след *else* ще се изпълни, ако условието е False. В примера по-долу нека създадем проста игра-викторина, в която от потребителя се иска да въведе столицата на Австралия. Ако отговорът е верен, ще покажем "Correct!" ("Вярно"). Ако отговорът не е верен, ще покажем "Incorrect. Try again" (Невярно. Опитай отново). Опитайте сами да въведете правилния отговор. По-долу ще покажем какво ще се случи, ако дадем неправилен отговор.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



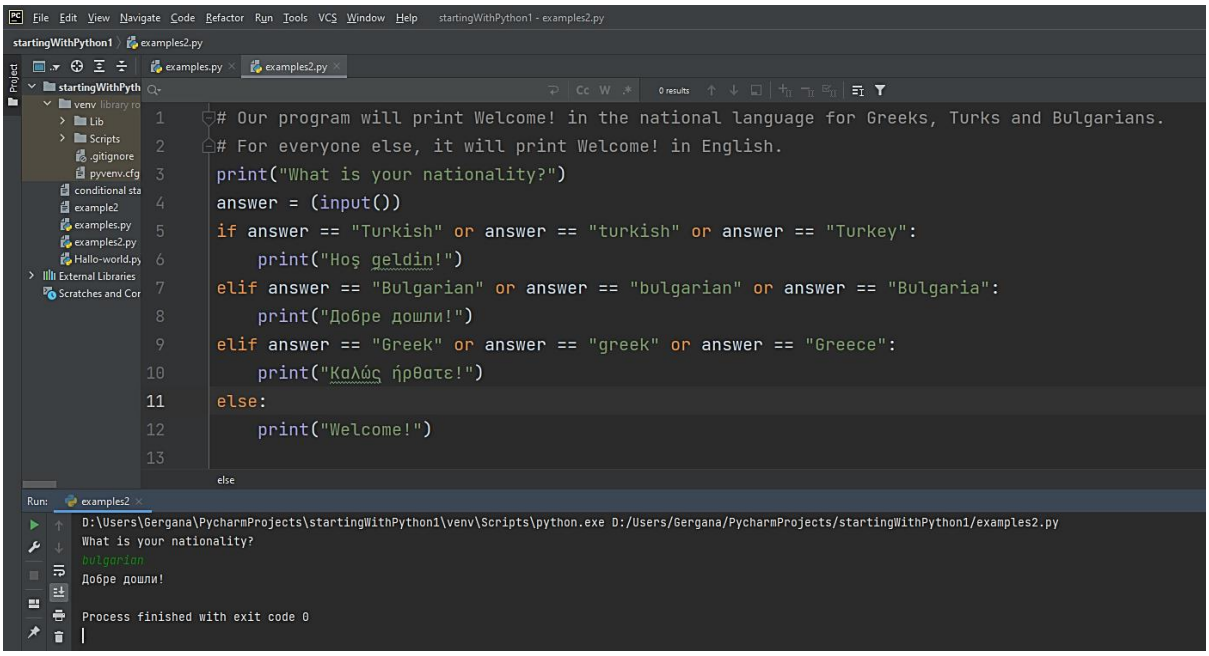
Разбира се, в по-сложните програми ще имаме по-сложна логика и ще трябва да изпълняваме различен код въз основа на различни входни данни. В тези случаи трябва да използваме конструкцията if-elif-else. Декларацията elif ни позволява да добавяме толкова алтернативни условия, колкото ни е необходимо. Когато едно от тях е True, се изпълнява определен код и условията в следващи декларации elif няма да бъдат проверявани. И отново, ако нито едно от условията elif не е вярно, посочваме какъв код ще бъде изпълнен с помощта на оператора else. Да кажем, че искаме определен текст да се показва само на хора с определена националност - ще напишем на конзолата “Добре дошли!” на различни езици в зависимост от националността на потребителя. Вижте в кода по-долу как реализираме тази логика.



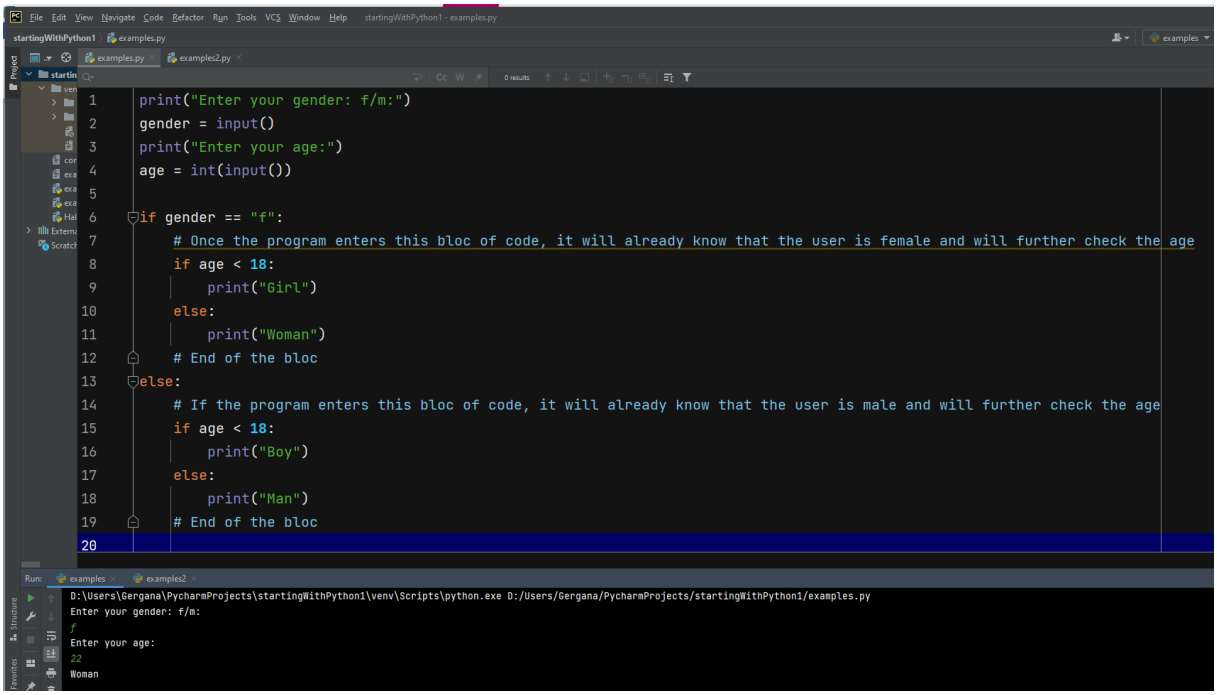
Можем да настроим програмата така, че да разпознава националността дори ако някой я напише само с малки букви или вместо нея напише името на държавата. За целта използваме оператора or, когато преценяваме дали условието е True или False. Можем да позволим условието да се оценява като True в различни случаи. В нашия пример указваме, че програмата ще напише *Добре дошли!* на български език, ако потребителят въведе или “Bulgarian”, или “bulgarian”, или “Bulgaria”.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



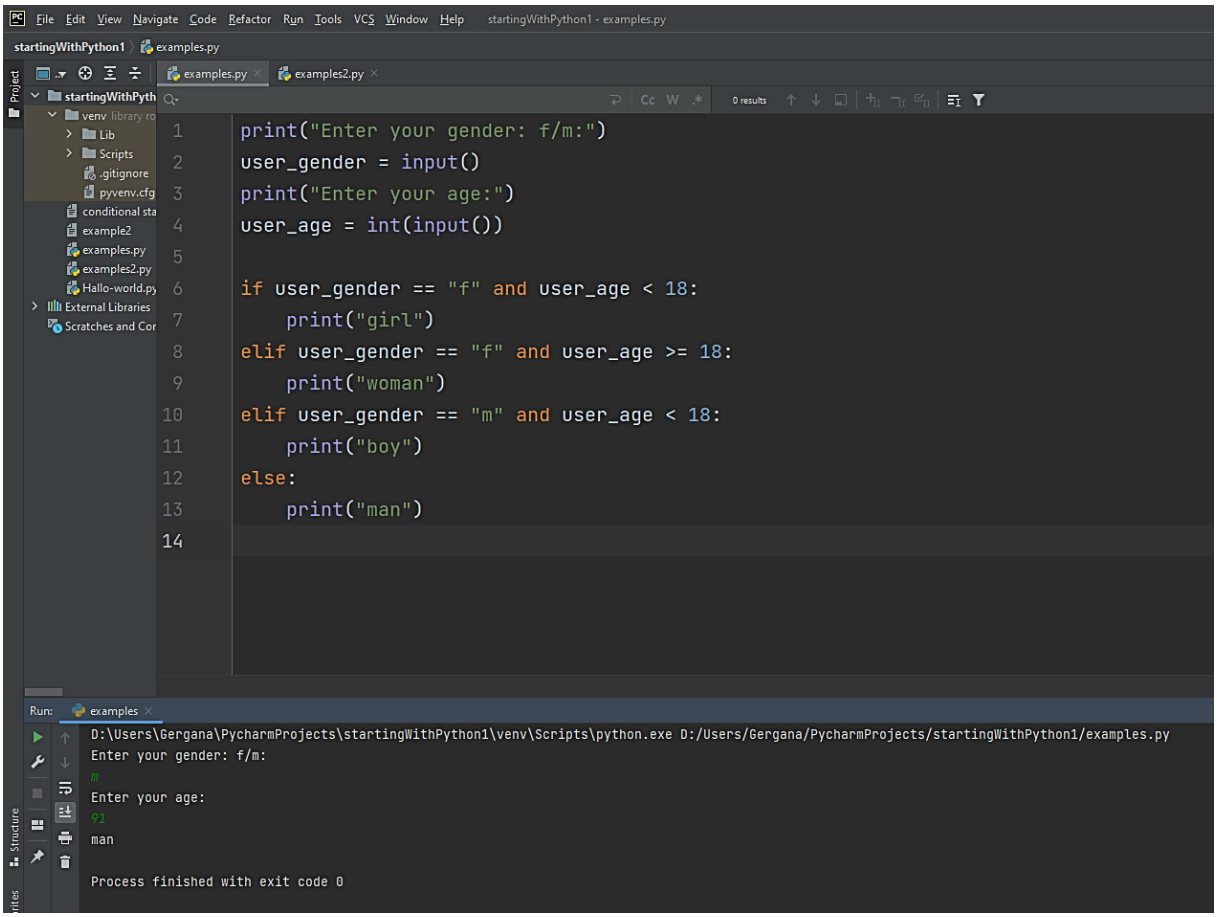
В някои случаи трябва да разработваме още по-сложни алгоритми, които да вземат решения на няколко нива. За да постигнем такава сложност, трябва да използваме вложени if-else оператори. Да кажем, че искаме потребителят да въведе пола и възрастта си и на тази база ще решим дали е "момче", "момиче", "жена" или "мъж". Първо ще проверим дали потребителят е жена или мъж, а след това в рамките на всяка опция ще проверим допълнително възрастта. Вижте кода по-долу, за да разберете как създаваме вложени *if-else* оператори и как програмата решава кой код да изпълни.



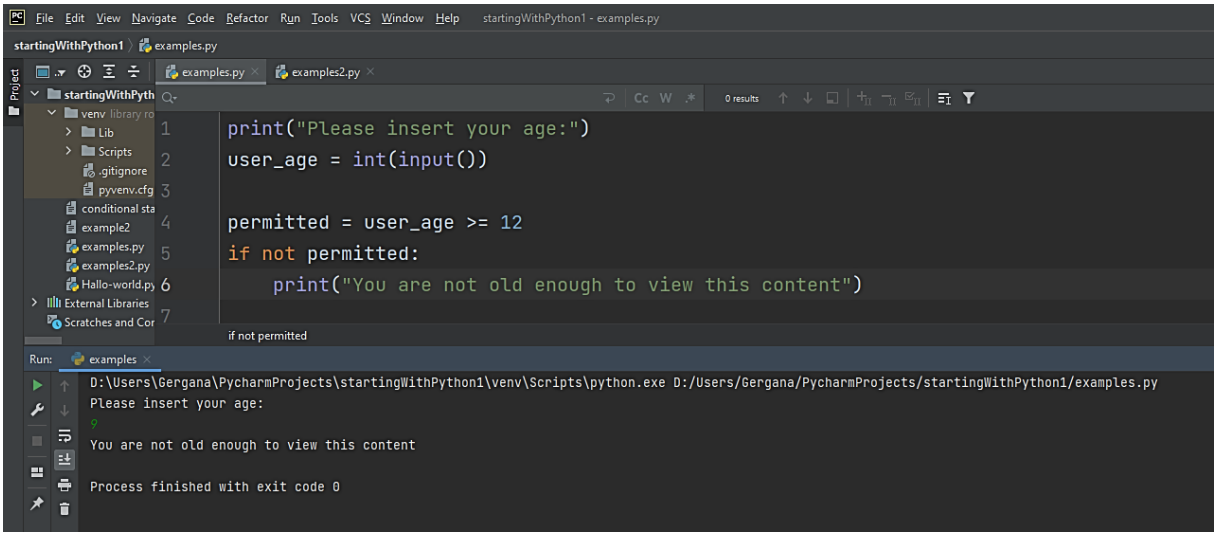
Разбира се, можем да постигнем същото и без вложени if-else оператори, като използваме if-elif оператора и комбинираме условията с оператора and. Показали сме този подход в кода по-долу. Изборът на единия или другия вариант зависи изцяло от личните ви предпочитания.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Във всеки случай трябва да се научите да използвате логическите оператори and, or и not. Операторът and връща True, ако и двете условия са True. Операторът or връща True, ако поне едно от условията е вярно. Операторът not ще върне True, ако условието е False. За да видите пример за използването на оператора or, вижте предишния пример за написването на "Добре дошли!" на различни езици. Нека дадем и пример за оператора not. Нека напишем програма, която ще пита за възрастта на потребителите и няма да им позволи да продължат да използват програмата, ако не са поне на 12 години.

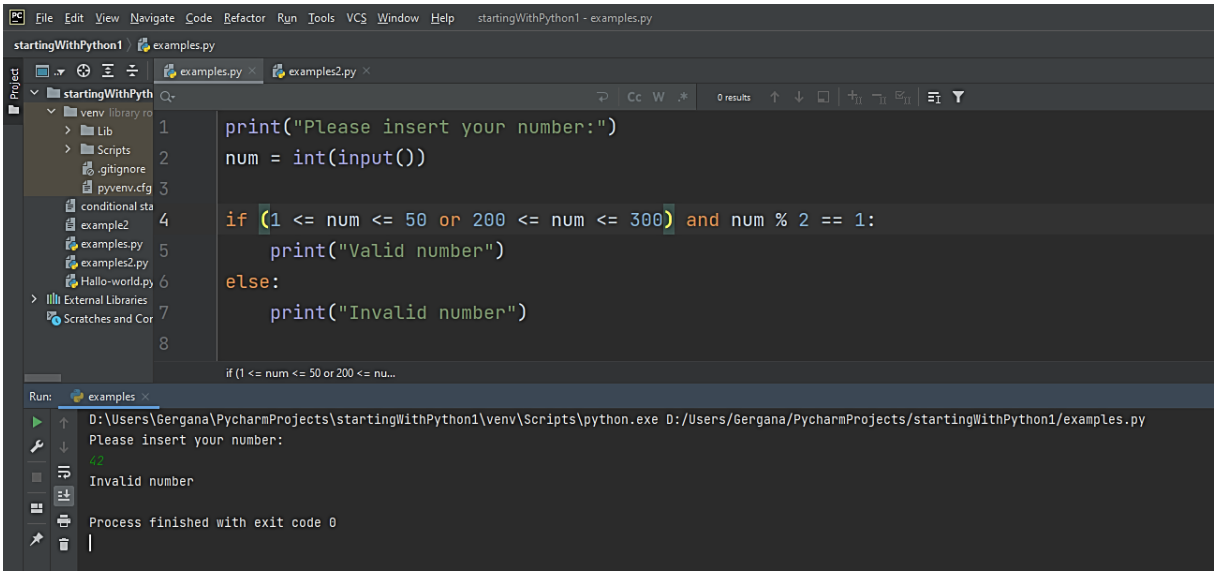


В някои случаи ще ни е необходима много сложна логика, когато някои условия трябва да се комбинират чрез използване на няколко оператора и and, or или not. Когато това е

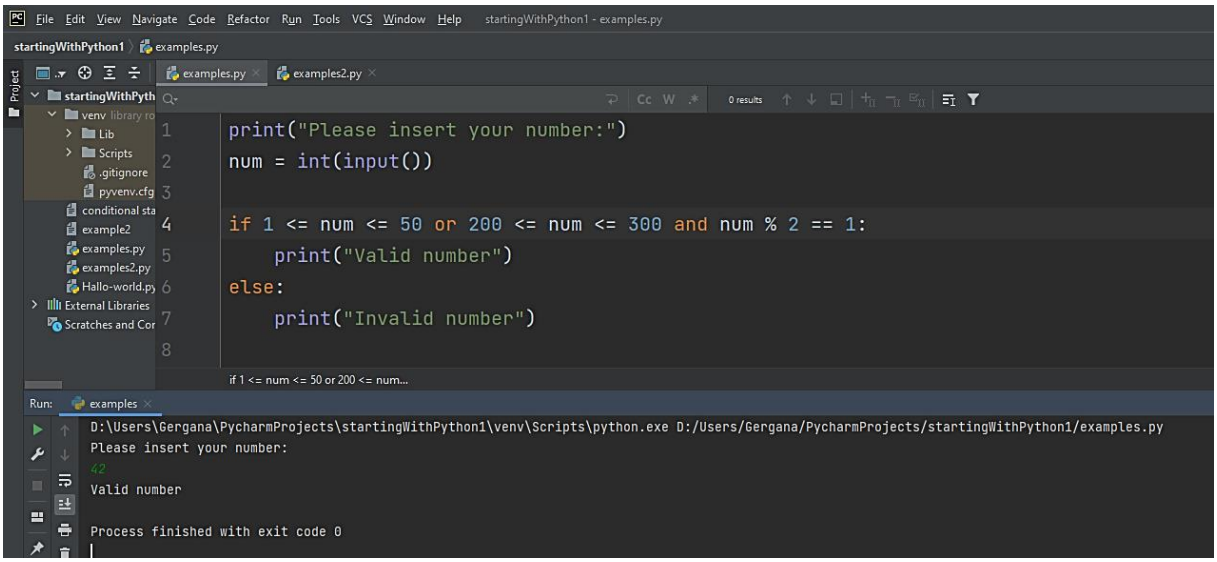
Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

необходимо за правилното изпълнение на логиката, се налага да комбинираме определени условия, поставяйки ги в скоби (), за да определим редът, в който трябва да се извърши оценката. В примера по-долу искаме да проверим дали дадено число е валидно. Определяме валидно число като нечетно число, което е между 1 и 50 или между 200 и 300. За да напишем този код, първо проверяваме дали а) числото е между 1 и 50 или между 200 и 300, а след това проверяваме дали б) то е нечетно число (за нечетно число модулното делене на 2 ще върне 1). Самото първо условие включва две условия, затова ще ги капсулираме в скоби, преди да използваме оператора and.



Забележете, че ако пропуснете скобите (вижте кода по-долу), резултатът за същото число (42) ще бъде грешен. Това е така, защото Python няма да разбере логиката на сравнението и ще оцени 42 като валидно, само защото попада в диапазона 1-50. Той ще провери дали числото е нечетно, само ако то е в диапазона 200-300. Често скобите са необходими за правилното изпълнение на логиката и обикновено подобряват четимостта на по-сложен код (дори и да не са необходими).



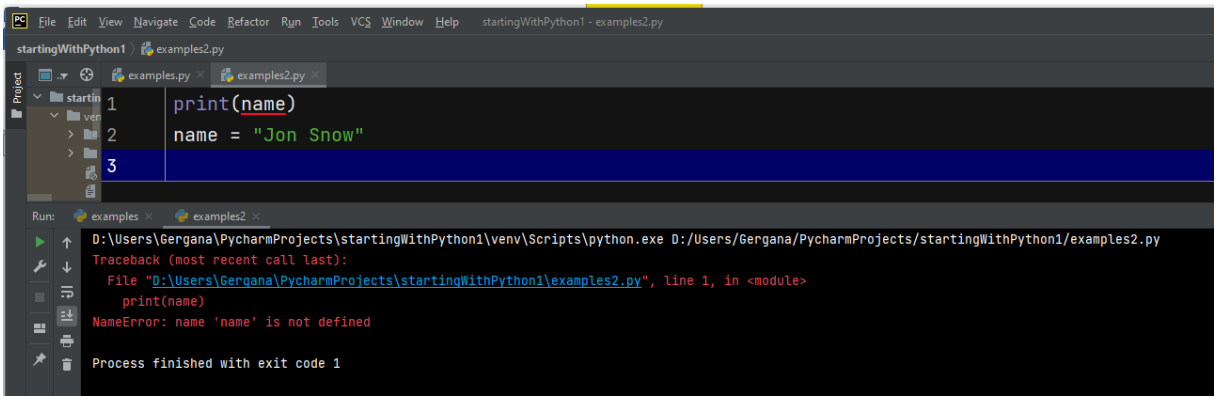
Вече сте готови да се тествате в HackerRank. Решете предизвикателството Python If-Else (уверете се, че сте избрали ниво на трудност "Easy").

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

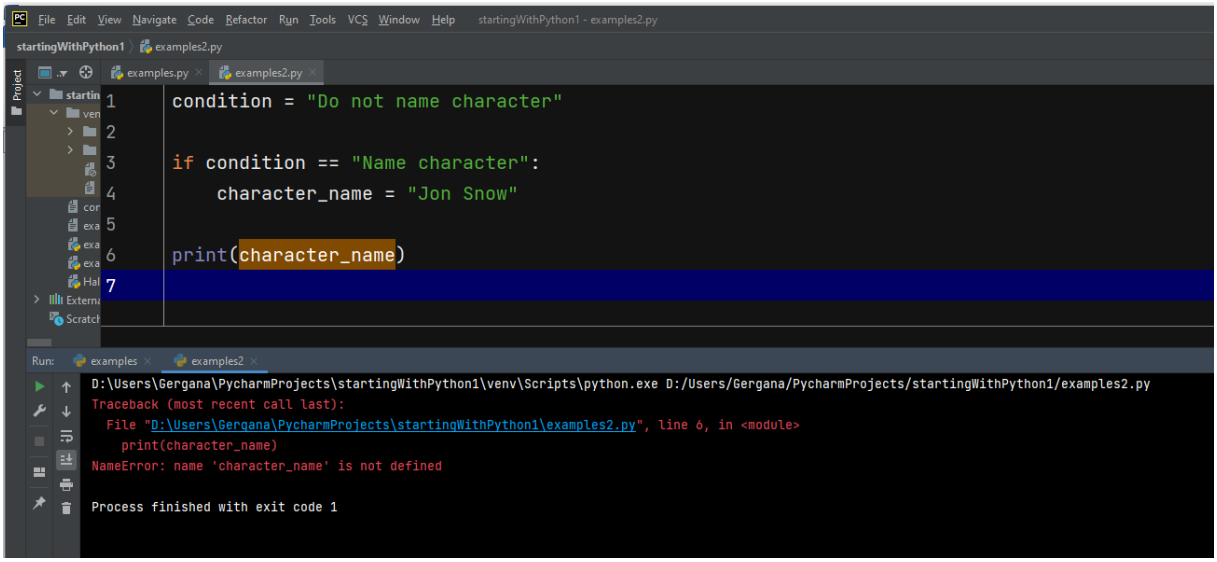
Инициализация и живот на променливите в Python; глобални и локални променливи

Променливите съществуват само ако са инициализирани някъде в програмата. В Python променливите могат да бъдат декларирани/инициализирани *извън функция* или *вътре във функция*. Това определя дали те имат глобален или локален обхват. Променливите, които са декларирани извън функция (в основната програма), са *глобални* променливи и са достъпни от всяка точка на програмата. Трябва да се уверите, че променливите са декларирани *преди да* бъдат използвани. Ако се опитате да използвате променлива, която е декларирана по-късно в програмата, ще получите грешка и програмата ще бъде прекратена.



Глобалните променливи могат да се променят в рамките на дадена функция, но за целта трябва да ги дефинирате с ключовата дума *global*.

Променливите, които са декларирани *в рамките на дадена функция,* са *локални* променливи. Те могат да се достъпват и използват само в рамките на функцията и се изтриват от паметта (престават да съществуват) след изпълнението на функцията. Това важи и за променливите, които са дефинирани в рамките на условни оператори. В примера по-долу не можем да отпечатаме name, защото тази променлива ще бъде дефинирана само ако изпълним оператора if, което не правим (защото условието е False); следователно променливата name никога не е инициализирана. Вашата ИСР ви предупреждава за възможността променливата ви да не е инициализирана, като я маркира в оранжево.



Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

Цикли в Python

В програмите много често се налага един и същ код да се повтаря отново и отново с различни входни данни. Например, може да имаме списък с имена и да се наложи да преминем през всички тях, за да проверим дали някое от тях е "Jon Snow". Вместо да пишем един и същ код 5 или 5000 пъти, ще използваме *цикъл*. Цикълът принуждава програмата да изпълнява затворения в него код толкова пъти, колкото ни е необходимо.

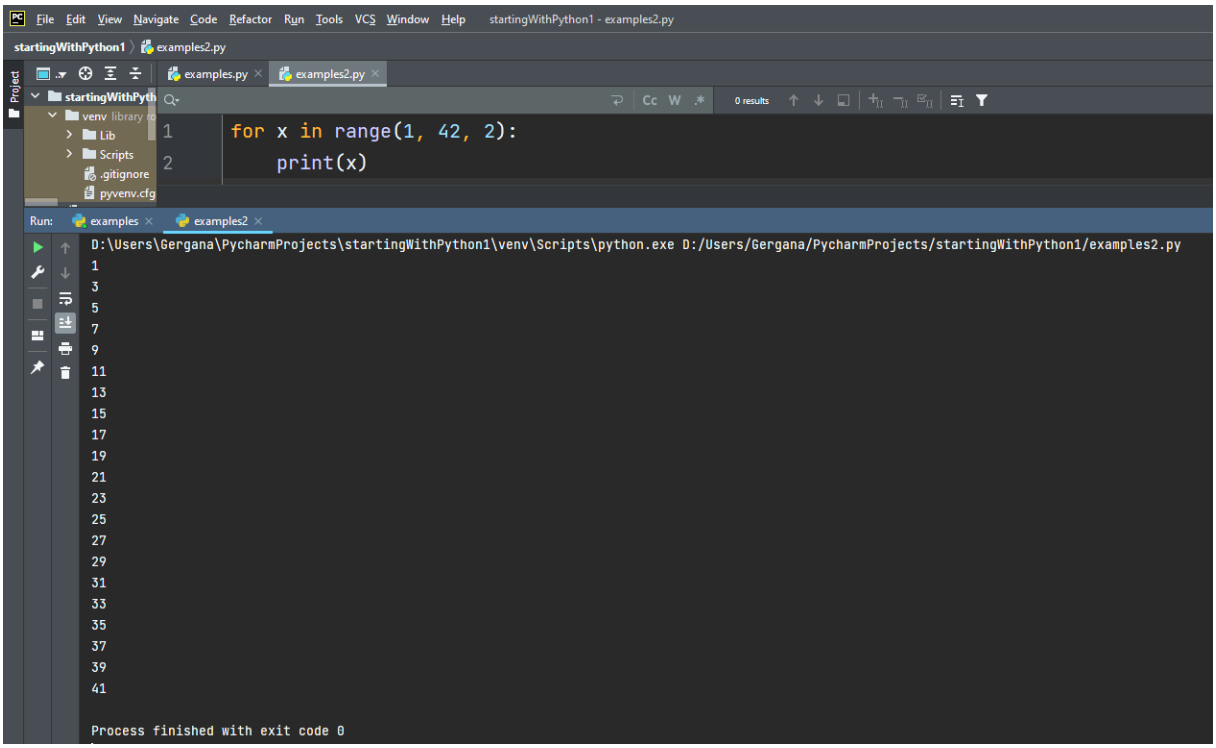
For цикъл

Цикълът *for* е първият основен тип цикъл. Той се конструира с помощта на *диапазон*, който определя колко пъти кодът трябва да бъде изпълнен отново (итериран). Пример за това е:

```
for x in range(1, 43): print(x)
```

Този код ще отпечата числата от 1 до 42 на нов ред, без да е необходимо да изписваме print 42 пъти. Забележете, че краят на диапазона е изключен, така че за диапазона 1-43 последното число, което ще отпечатаме, е 42. Опитайте сами в PyCharm.

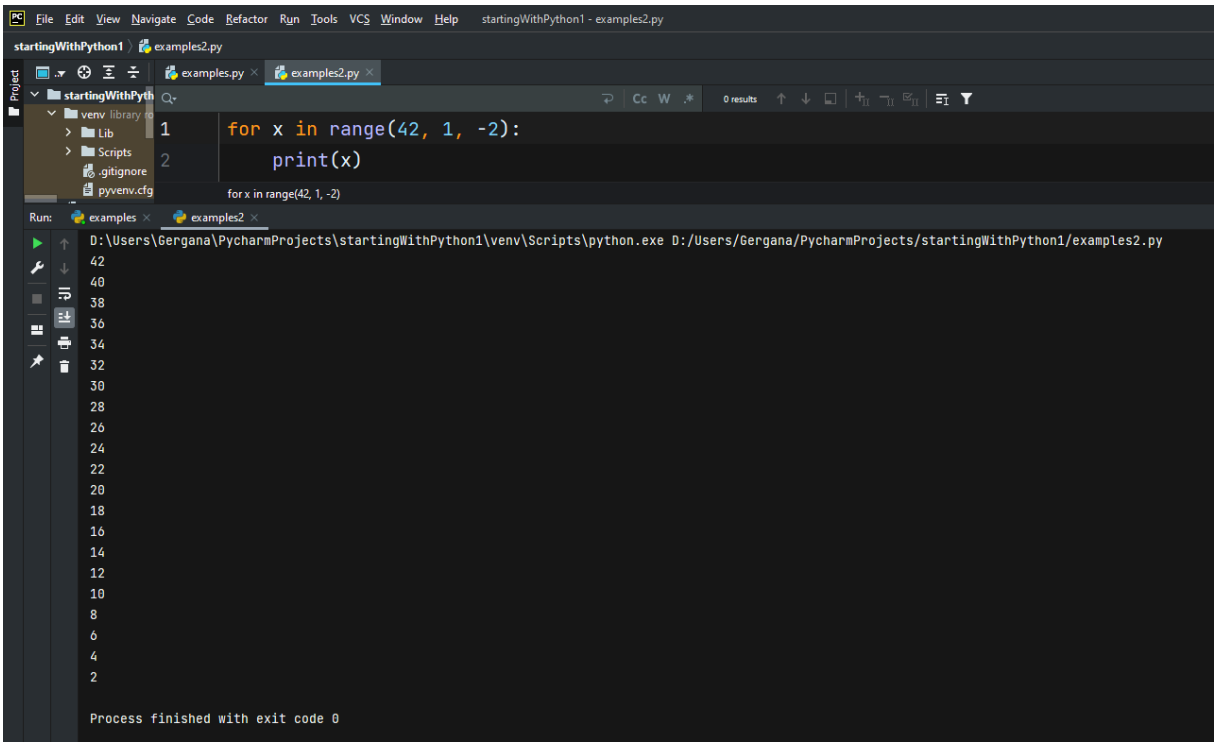
Можем да зададем и стъпка за итерацията. Ако искаме да отпечатваме само всяко второ число от интервала 1 - 42 (т.е. само нечетните числа), ще добавим *стъпка* 2 при конструирането на цикъла. Добавяме я след запетая, следваща диапазона. Вижте този код по-долу.



Ако итерираме в обратна посока, можем да добавим отрицателна стъпка от -1 или друга отрицателна стъпка. Вижте такъв цикъл по-долу, с отрицателна стъпка -2.

Съфинансиран от програма „Еразъм+" на Европейския съюз

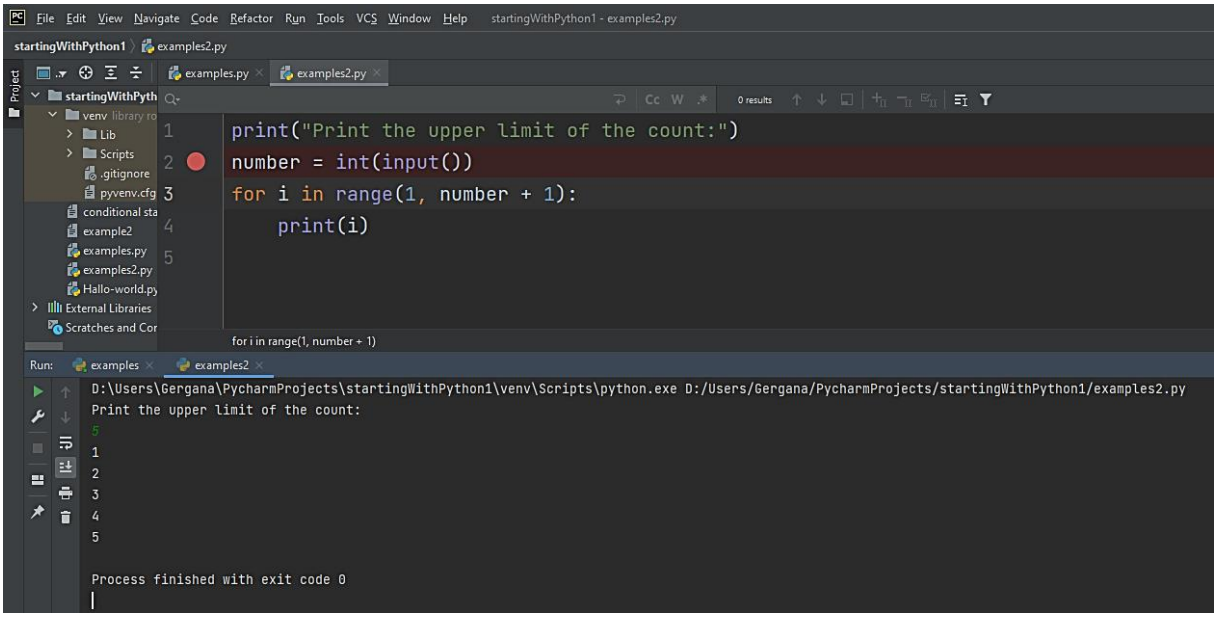
Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Най-често няма да знаем колко пъти трябва да итерираме. Броят на итерациите ще бъде зададен от променлива в нашата програма или от потребителя. Затова е възможно да определим обхвата на цикъла с променлива, която ще се променя или подава от програмата. В примера по-долу отпечатваме числата от 1 до числото, което потребителят въведе на конзолата. Забележете, че горната граница на обхвата е числото на потребителя + 1, защото краят на диапазона е изключен.

Съвет за именуване на променливи в цикъла:

Типичната буква, която използваме за итерация в един цикъл, е *i*, а не *x*, която използвахме в досегашните примери. Програмата обаче ще работи с всяка буква или дума, която изберете.



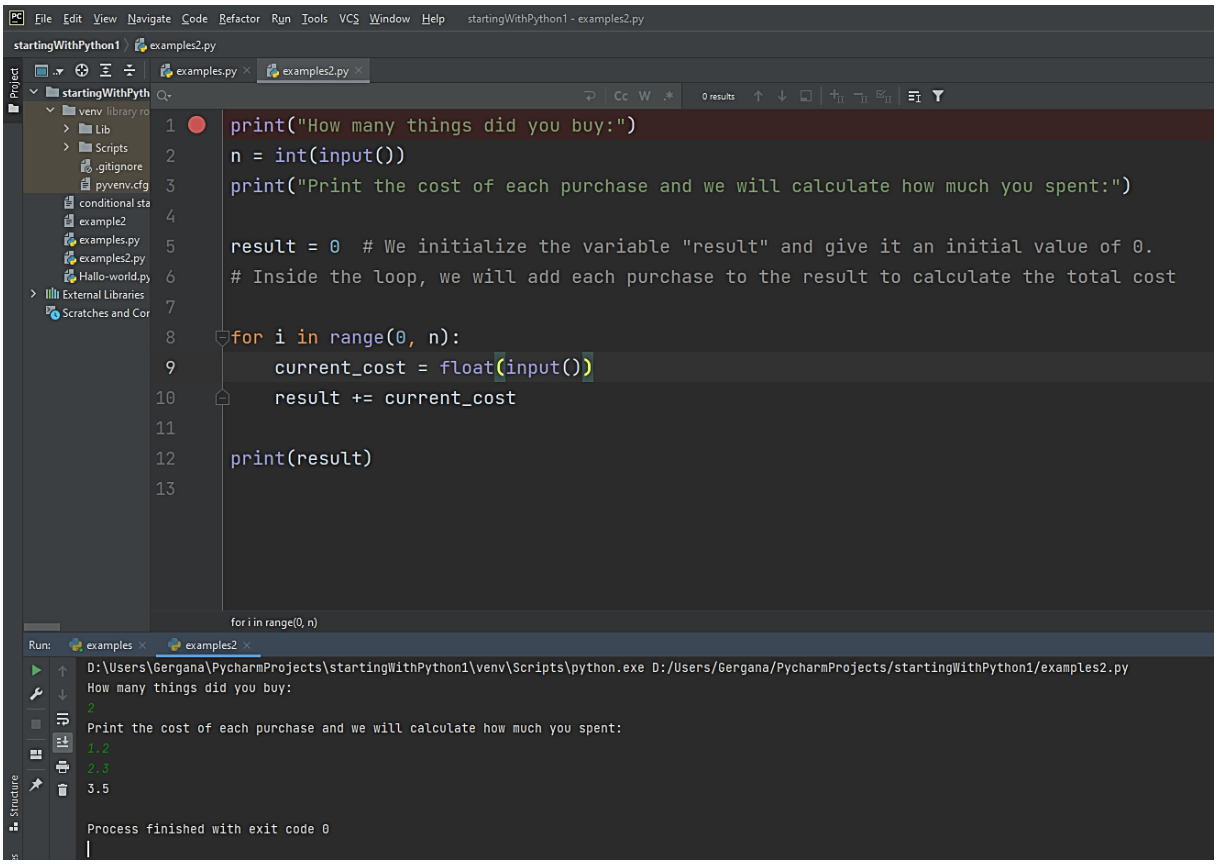
Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

Най-хубавото нещо на *for* циклите (а всъщност и на всички видове цикли) е, че в тях можем да пишем много сложен код. Това ни позволява да изпълняваме много сложна логика многократно, но с много малко код за писане.

Примери за използване на *for* цикли

Нека позволим на потребителя да изчисли разходите си, като ги запише един след друг. Ще използваме *for* цикъл, за да съберем всички разходи, и ще върнем на конзолата общата стойност. Не забравяйте да инициализирате променливата, представляваща общите разходи, извън цикъла, за да можете да я отпечатате по-късно. Обърнете внимание, че започнахме цикъла от 0, вместо от 1 (това е по-конвенционално), така че краят на обхвата на цикъла ще бъде n, а не n+1. Както range(1, n+1), така и range(0, n) ще дадат един и същ резултат.

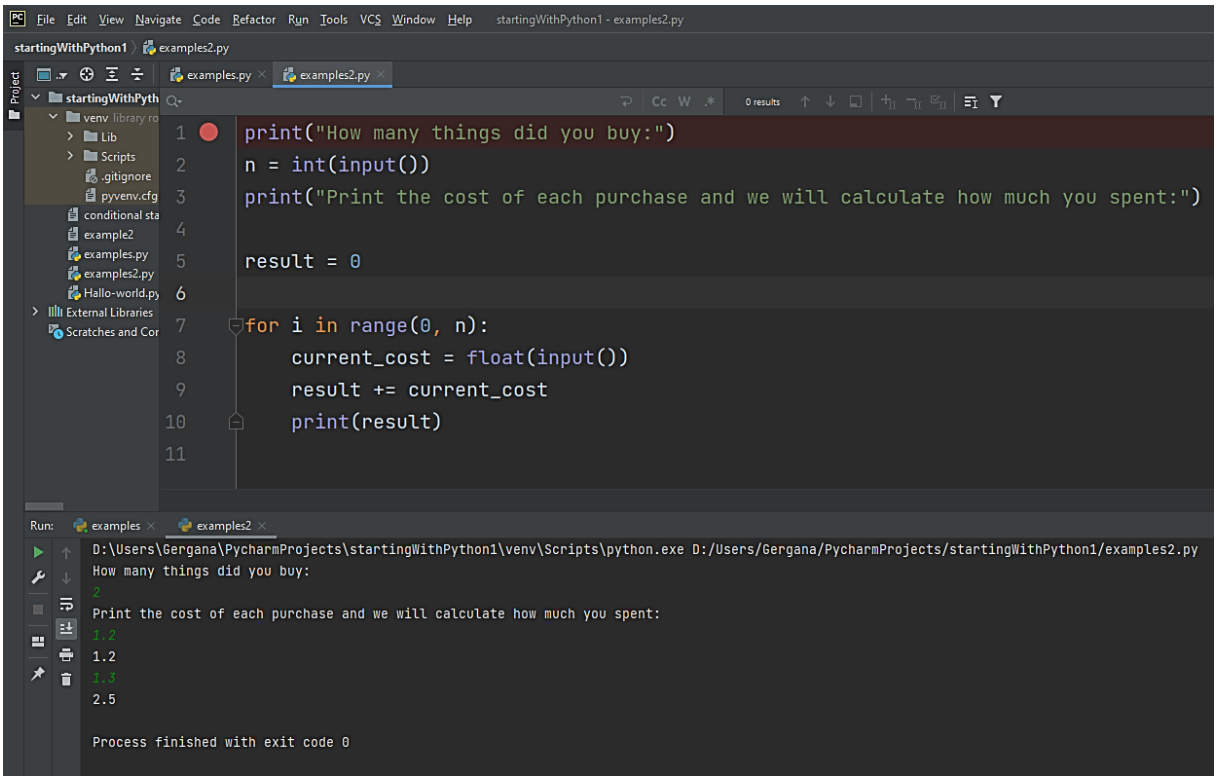


Съвет за разграничаване на кода на цикъла от другия код:

Отстъпът показва на Python дали дадена команда е в цикъла или извън него. В други езици за програмиране, с които може би сте запознати, кодът на цикъла се поставя в къдрави скоби (напр. в Java). Но в Python трябва да се обърне специално внимание на отстъпването. В горния пример print(result) не е отстъпен и следователно отпечатването ще бъде извършено, след като програмата излезе от цикъла. Ако го отстъпите до нивото на командите вътре в цикъла (както в примера по-долу), програмата ще отпечатва резултата при всяка итерация.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

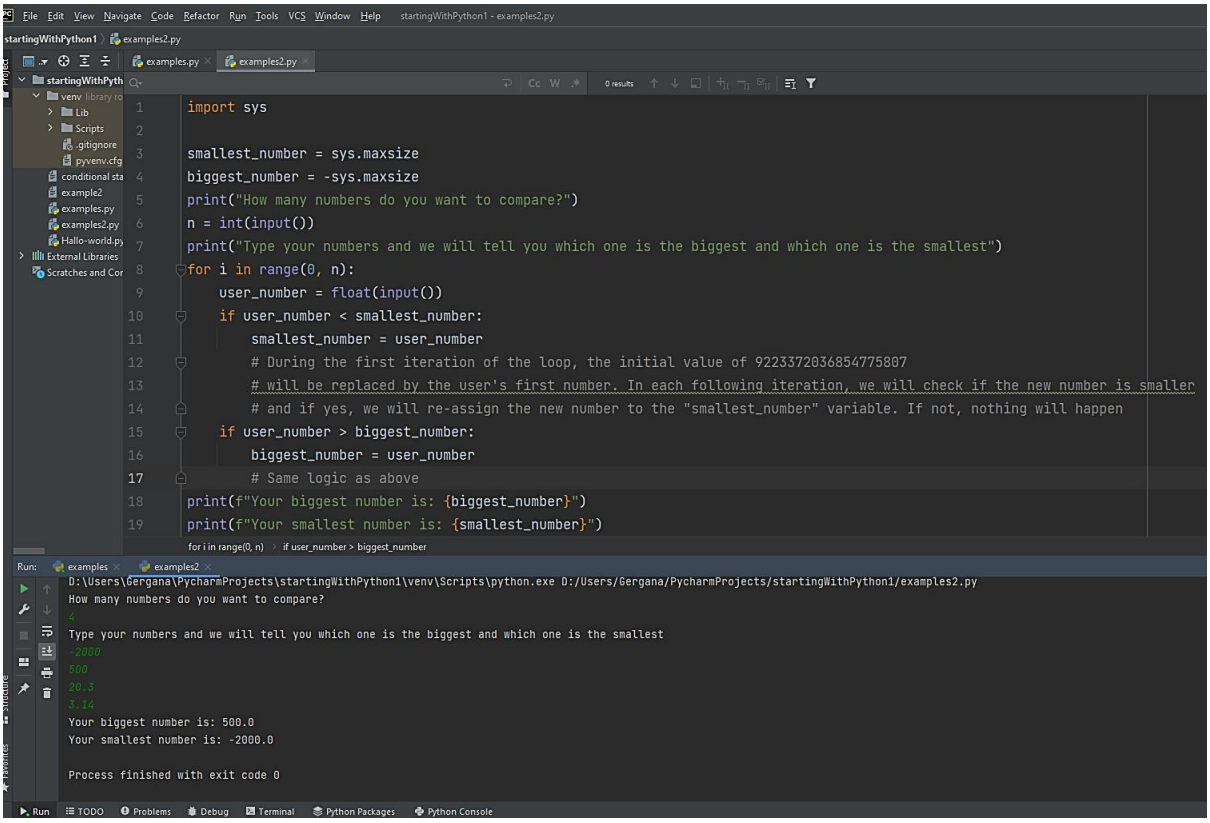
Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Нека сега да напишем програма, която позволява на потребителя да въведе толкова числа, колкото желае, и програмата ще изведе най-голямото и най-малкото число. Програмата ще приема както дробни числа, така и цели числа, а също и положителни и отрицателни числа. Специфичното тук е как да инициализираме променливите, които са ни необходими. Очевидно се нуждаем от променлива за най-голямото число и от променлива за най-малкото число, като те трябва да бъдат инициализирани преди цикъла (извън него). Но как да зададем началната стойност на всяка от тях? Ако погледнете кода по-долу, ще забележите, че всяко число, въведено от потребителя, се сравнява с най-голямото и най-малкото число в момента и ако то е съответно по-голямо от най-голямото или по-малко от най-малкото, тогава самото текущо число ще стане съответно най-голямото или най-малкото. Следователно, когато инициализираме числата, първоначално най-голямото число трябва да бъде *най-малкото* възможно число; в противен случай зададеното от потребителя число може да се окаже по-малко от него и ние няма да заменим тази първоначална най-голяма стойност. По същия начин най-малкото число трябва първоначално да бъде *най-голямото* възможно число. По този начин няма да е възможно да пропуснем което и да е дефинирано от потребителя число, което е по-малко от първоначалната стойност. В Python най-голямото възможно число в 64-битова система е 9223372036854775807, а най-малкото е -9223372036854775807. За 32-битови системи най-голямото число е 2147483647, а най-малкото е -2147483647. Тези стойности се дават от функцията sys.maxsize() на Python. За да използваме тази функция, трябва да импортираме библиотеката sys. Вижте целия код по-долу.

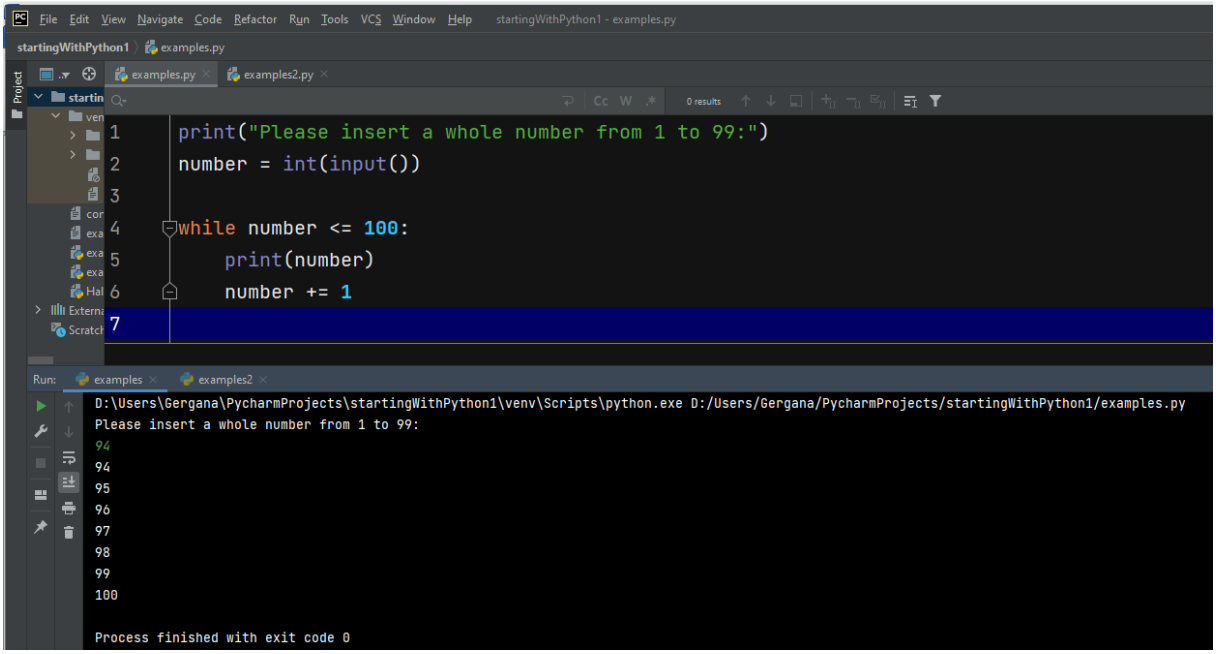
Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



While цикъл

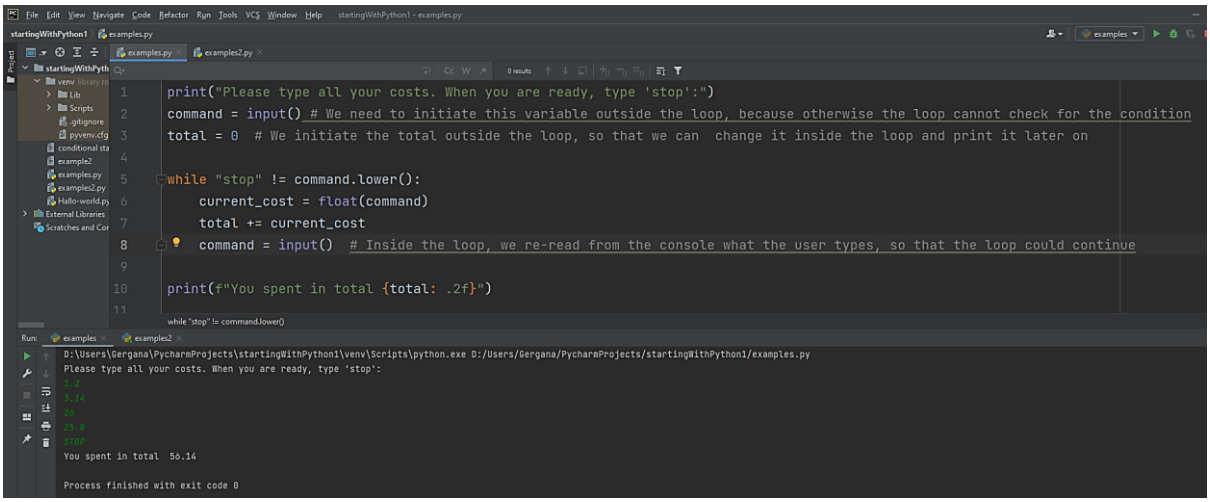
For цикълът изисква от нас да зададем точния диапазон за итерация. Но какво да правим, ако не знаем диапазона или просто искаме да итерираме, докато е валидно някакво условие? За тази цел използваме *while* цикъл. *While* цикълът задава условие, което може да бъде оценено като True или False, и итерира, докато това условие е True. Например, нека помолим потребителя да въведе число от 1 до 99, след което да изведем всички числа, по-големи от числото на потребителя, докато стигнем до 100. В кода по-долу ще забележите, че повтаряме функцията print(), докато числото стане равно на 100, и при всяка итерация увеличаваме началното число с 1.



Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

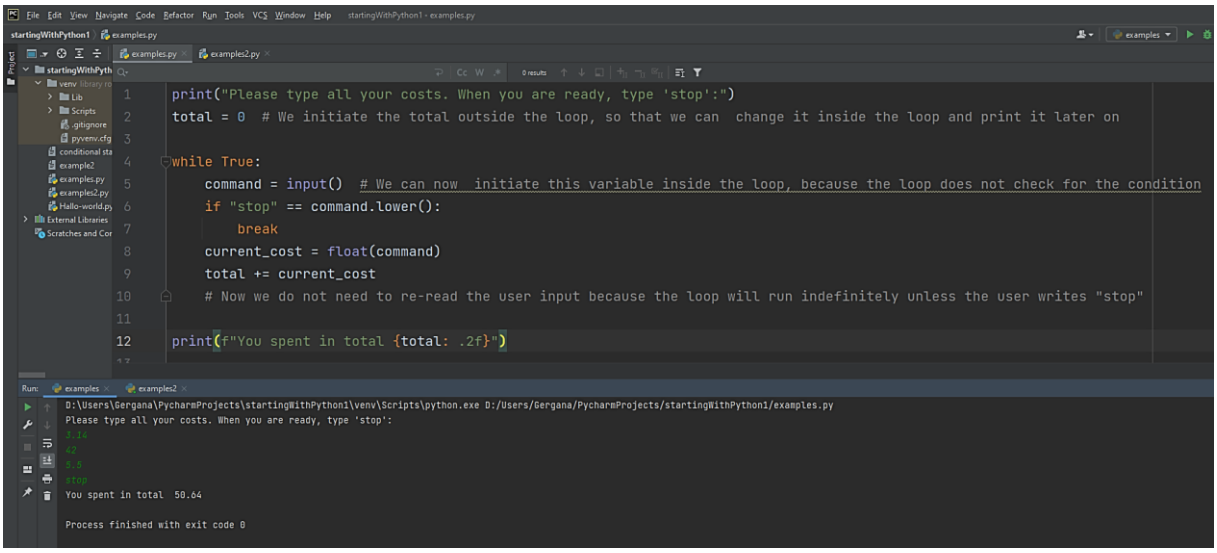
Нека сега използваме *while* цикъл, за да пренапишем предишната програма, в която потребителят въвежда разходите си, а ние изчисляваме общата сума на похарчените пари. В предишната програма помолихме потребителите да кажат колко разходи ще въведат и използвахме *for* цикъл с диапазон, определен от отговора на потребителя. Сега ще оставим потребителите да въвеждат разходи, докато не напишат "Стоп". След това ще изчислим общите разходи и ще ги отпечатаме на конзолата. Вижте кода по-долу. Прочетете коментарите за инициализиране на променливите. Обърнете внимание, че първо прочитаме потребителския вход извън цикъла, за да разберем дали изобщо трябва да влизаме в цикъла. След това вътре в цикъла го препрочитаме при всяка итерация, за да проверим дали условието на цикъла все още е валидно за следващата итерация. Всеки път, когато препрочитаме входа, на променливата command се присвоява новият вход като стойност. Трябва също да забележите, че първо четем входа като string, а го превръщаме в десетична дроб едва след като влезем в цикъла. Това е така, защото, ако го превърнем веднага, щом потребителят напише "stop", програмата няма да може да го превърне и ще генерира грешка. Проверяваме дали командата е "Stop" или "Stop" или "STOP". Правим това, като сравняваме двата низа - низа на command и низа "stop" - по начин, при който не се отчитат големи и малки букви. Използваме метода *my_string*.lower(), за да преобразуваме command само в малки букви. По този начин, въпреки че входа в нашия пример беше "STOP" вместо "stop", програмата беше изпълнена правилно.



Разновидност на *while* цикъла е *while True* цикълът. Този цикъл ще работи безкрайно дълго, докато не се появи команда, която да го прекъсне - командата break. Можем да извикаме тази команда в условна конструкция, така че цикълът да се прекъсне, когато определено условие е True. Същото обаче може лесно да се постигне с вмъкване на условието в самия *while* цикъл, така че използването на *while True* цикъла обикновено не е препоръчителна практика (може да затрудни промяната на кода). Все пак има някои случаи, когато е оправдано използването на *while True* цикъл, като например в случая на HTTP слушател, където наистина се нуждаем от безкраен цикъл. Във всеки случай, по-долу можете да видите горния пример, реализиран с цикъл *while True*.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Вече сте готови да се тествате в HackerRank. Решете предизвикателството “Loops”. То е достъпно в режим на трудност “Easy”. Опитайте и предизвикателството “Print Function”, което ще провери знанията ви както за отпечатването на конзолата, така и за *for* циклите.

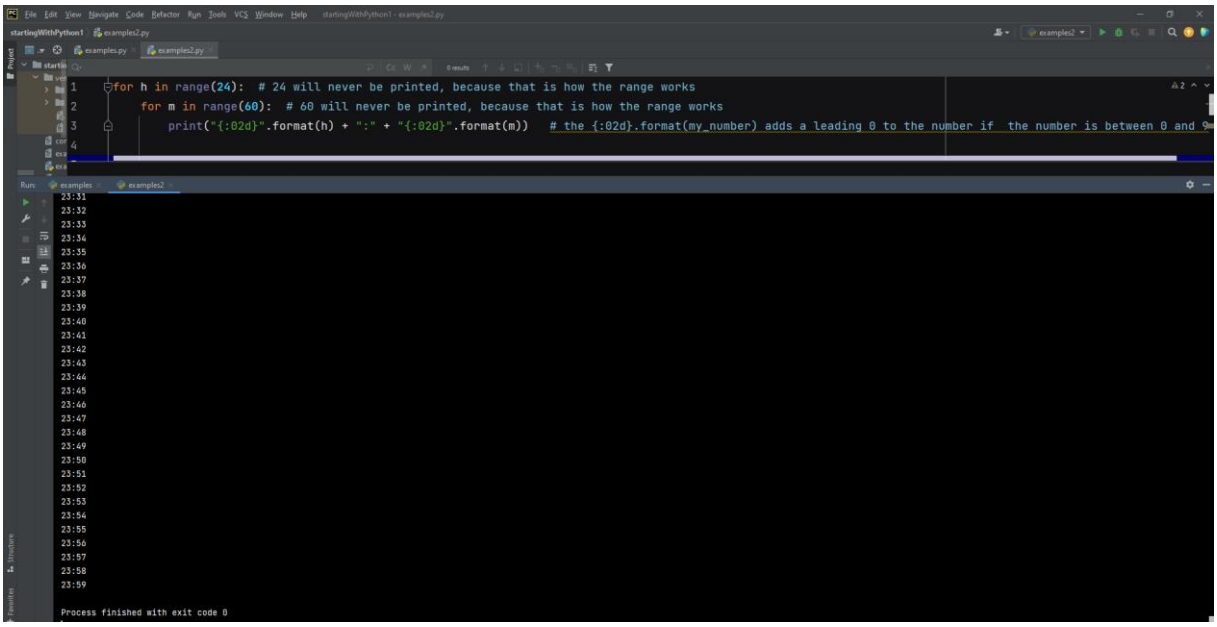
Вложени цикли

При по-сложни алгоритми е необходимо да се итерира по няколко колекции от данни едновременно. За тази цел са необходими вложени цикли. Итерацията започва от външния цикъл. След като се зареди първата стойност от външния цикъл, програмата итерира по всички стойности във вътрешния цикъл. След това се зарежда втората стойност от външния цикъл и програмата отново итерира по всички стойности във вътрешния цикъл. Това се повтаря, докато се премине през всички стойности във външния цикъл.

За да разберете вложените цикли, най-добре е да опитате един пример. Нека отпечатаме минутите и часовете на един ден, като използваме вложени цикли. Знаем колко часа има и колко минути има на час, така че можем да използваме вложени *for* цикли. Както знаете, трябва да минат 60 минути, преди да се смени часът. Така външният цикъл е за часовете, а вътрешният - за минутите.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Какво прави тази програма? Влиза във външния цикъл и взима първата стойност в диапазона 0-24, която е 0 (това е часът). След като тази стойност бъде заредена, програмата влиза във вътрешния цикъл и започва да итерира по всички стойности във вътрешния цикъл (минутите). Тя ще зареди всички стойности, започвайки от 0 и завършвайки с 59 (крайният индекс - таванът на диапазона - не е включен, спомняте си). При отпечатването сме добавили водеща нула, която да се отпечатва винаги, когато числото е между 0 и 9. Така вместо 1 ще отпечатаме 01. При всяка итерация на вътрешния цикъл програмата ще отпечатва часовете и минутите: 00:00, 00:01, 00:02.....00:59. След като програмата достигне 60 във вътрешния цикъл, тя ще излезе от вътрешния цикъл, ще влезе отново във външния цикъл и ще зареди втората стойност от външния цикъл, а именно 1. След това ще итерира отново по всички стойности във вътрешния цикъл (от 0 до 59) и ще отпечата: 01:00, 01:01, 01:02....01:59. След това отново ще излезе от вътрешния цикъл, ще влезе във външния цикъл и ще зареди третата стойност във външния цикъл, а именно 2. Това ще се повтори до последната итерация на външния цикъл, а именно 23. След като зареди 23, програмата ще итерира през 0-60 във вътрешния цикъл и ще отпечата: 23:00, 23:01, 23:02....23:59. Последната отпечатана стойност ще бъде 23 часа и 59 минути. Напишете програмата, за да проверите резултатите сами в PyCharm.

Съвет за по-ефективно писане на код:

Забележете, че не е необходимо да пишем range(0, 24). Написахме само range(24), тъй като това е еквивалентно, когато началната стойност е 0.

Съвет за форматиране:

Водещата нула се изписва с "{:02d}".format(*моето_число*). Тази функция добавя нула, ако числото съдържа само една цифра. Ако напишете "{:03d}".format(*моето_число*), ще получите 1 или 2 водещи нули, така че отпечатаното число винаги ще има 3 цифри.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

Работа с текст

Основни функции

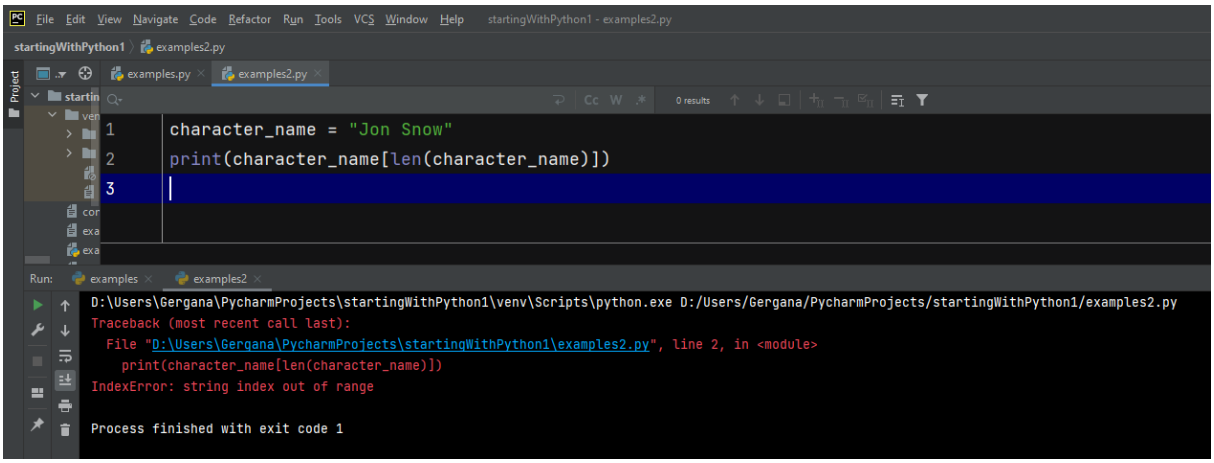
В програмите често ни се налага да работим с низове (текст). Python предлага много команди и функции, които ни позволяват да използваме и манипулираме низове, както желаем.

Функцията len(*текст*) връща дължината на даден низ (независимо дали е дума или дълъг текст). Изпробвайте я сами в PyCharm.

Функцията *текстова_променлива*[*i*] ще върне символа на i-тия индекс на променливата *текстова_променлива*. Индексът указва позицията на символа в низа. Например:

```
character_name = "Jon Snow"
print(character_name[0])
```

Това ще върне "J". Индексите са винаги цели числа. Първият индекс на всеки низ е винаги 0, така че последният индекс е *дължината на низа* - 1. Объркването на индексите ще доведе до грешка "index out of range".



Горният пример завърши с грешка. Опитахме се да получим последния индекс, като използвахме дължината на низа. Това е интуитивно, но грешно, защото започнахме от 0, а не от 1. Така че, ако искате да вземете последния символ на низа, трябва да използвате:

```
character_name = "Jon Snow"
print(character_name[len(character_name) - 1])
```

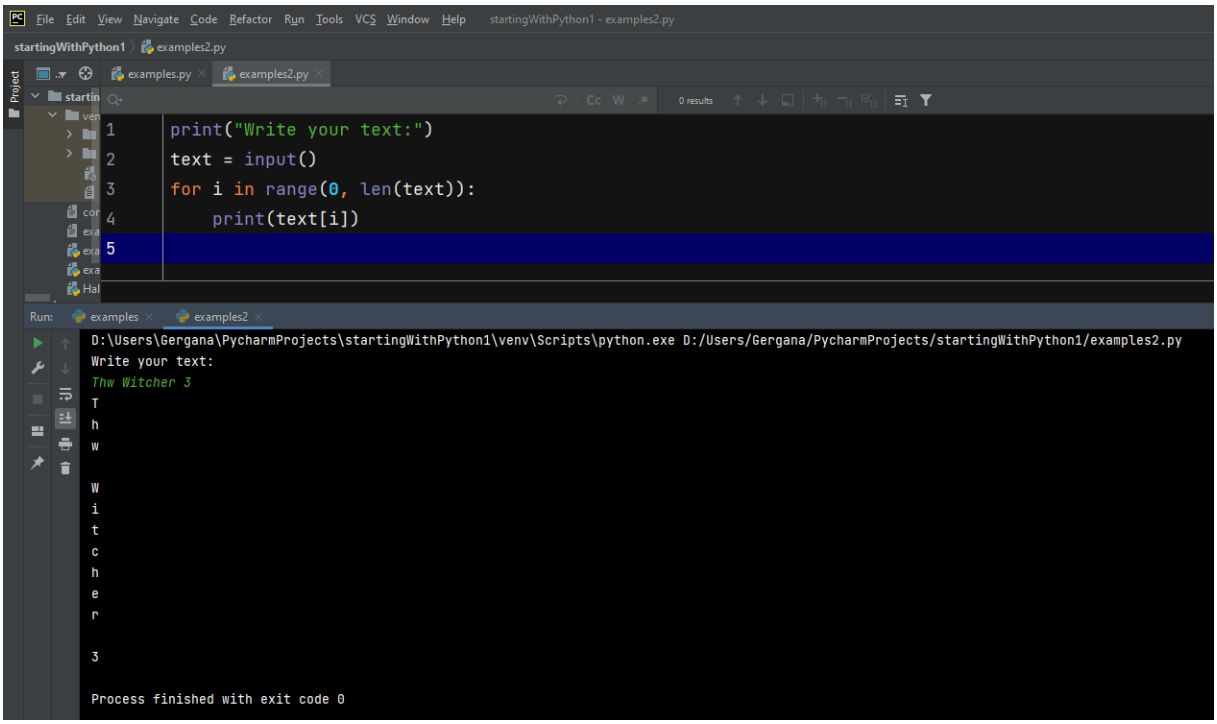
Можете да използвате *for* цикъл, за да прегледате всички символи в даден низ. Например, с кода по-долу записваме всеки символ от "Jon Snow" на нов ред. Забележете, че тъй като обхватът на цикъла не включва последния индекс, не изваждаме 1 от дължината, както направихме по-горе. Извършваме итерация точно до дължината на низа, защото знаем, че цикълът ще бъде прекратен преди последния индекс - len(*име_на_символа*) (който е извън обхват).

```
character_name = "Jon Snow"
for i in range(0, len(character_name)):
    print(character_name[i])
```

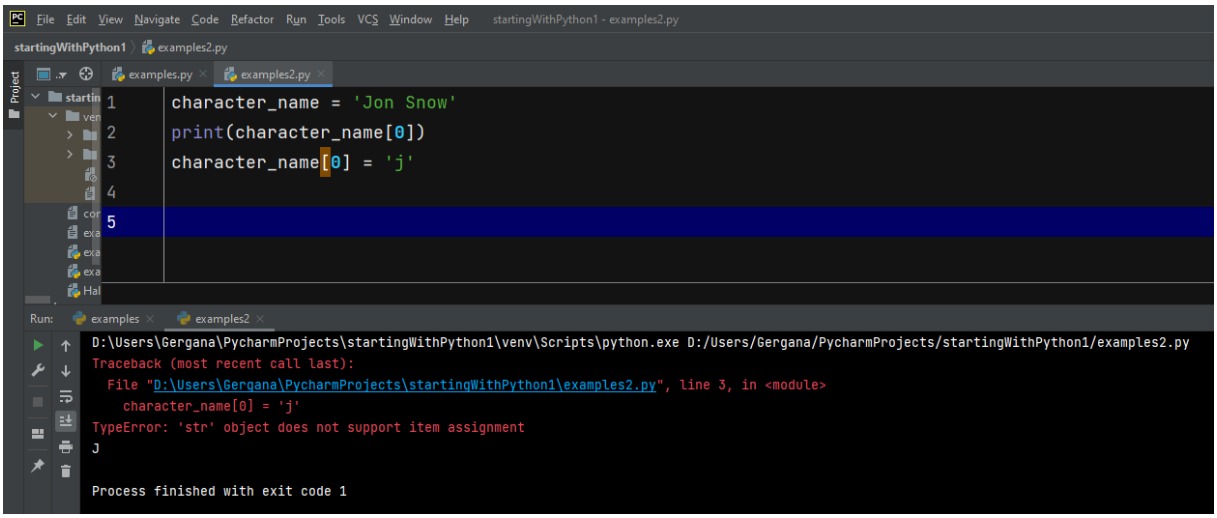
Същото можем да направим и за всеки текст, въведен от потребителя в конзолата:

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



В Python низовете и не могат да бъдат променяни. Можете да отпечатате определен символ, но не можете да го промените. В примера по-долу можем да отпечатаме първата буква от текста, но когато се опитаме да я променим, получаваме грешка.

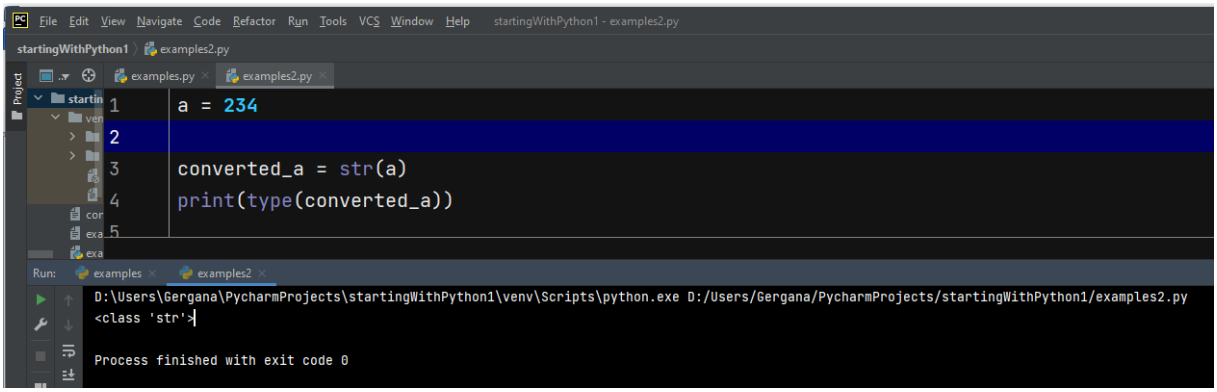


Повече за форматирането на низове

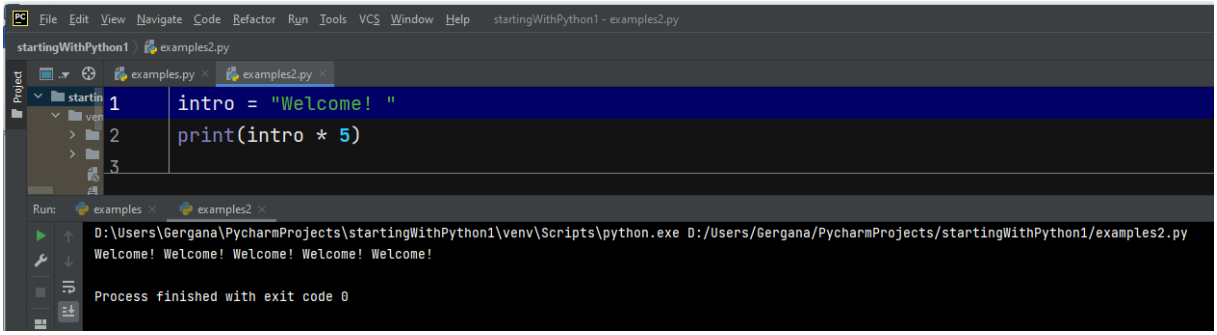
В предишните раздели вече научихме, че когато трябва да преобразуваме друг тип данни в низ, използваме функцията str() и поставяме променливата, която трябва да преобразуваме, в кръглите скоби на функцията (подаваме я като аргумент). Можем да използваме тази функция, за да преобразуваме в низ всеки вграден тип данни в Python, включително дробни числа, списъци, кортежи, речници и т.н. По-късно ще научим повече за някои от тези типове данни, а след това ще можете да изпробвате функцията. По-долу отново ще покажем прост пример за преобразуване на цяло число.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

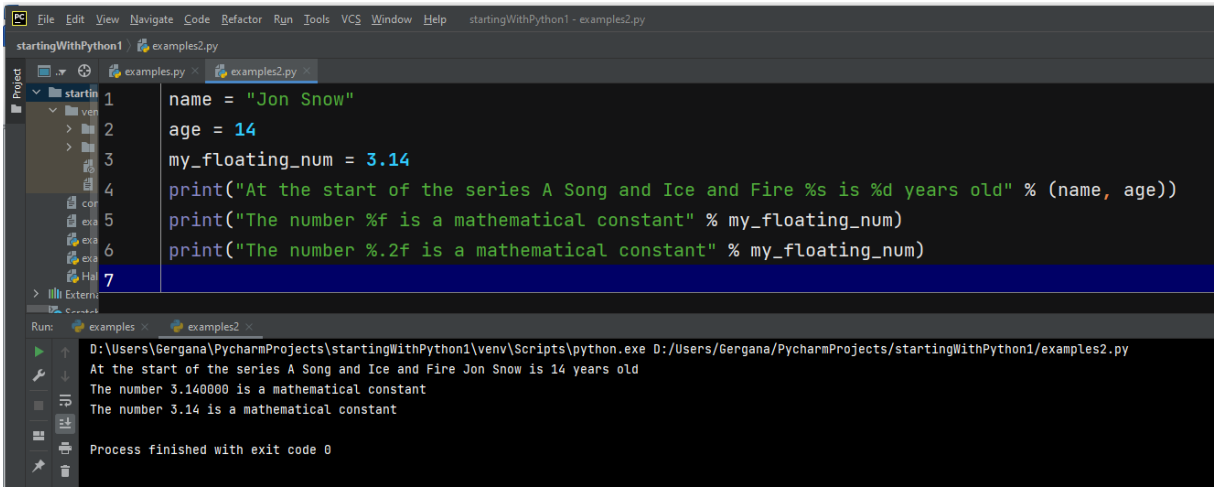
Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Можем да използваме оператора *, за да повторим даден низ определен брой пъти.



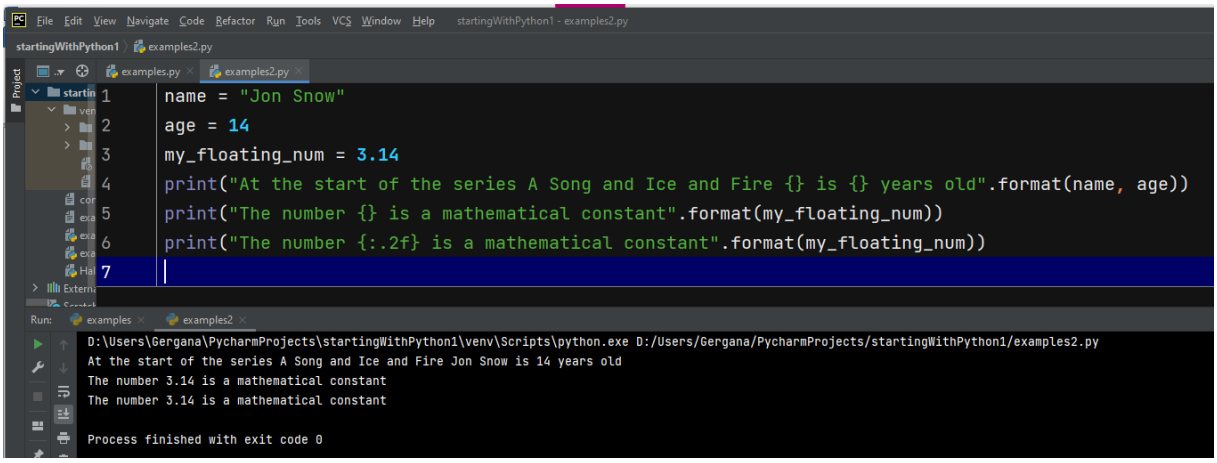
Сега нека се върнем към работата с текст, който включва стойностите на определени променливи (т.е. тази стойност не е предварително известна, а се предоставя по време на изпълнението на програмата). Справяме се с тази задача чрез форматиране на низове. Вече научихме за форматирането на низове с помощта на метода f "*форматиран низ*". Съществуват и други методи, които може да видите и използвате. Можете да форматирате с оператора %, последван от суфикса s за низ, суфикса d за цяло число и суфикса f за дробно число. Комбинацията от % и суфикс служи като заместител на променливата, която се подава след низа, като се използва синтаксиса % *моята_променлива*. Ако се използват повече заместители, се използва % (*моята_променлива_1, моята_променлива_2*). Вижте примерите по-долу. Суфиксът f позволява допълнително форматиране. Ако добавим .2 или .3 преди него, ще укажем колко цифри ще бъдат показани след десетичния знак на дробно число. Можем да посочим произволен брой цифри. Ако не посочим нищо, по подразбиране Python показва дробните числа до 6 цифри след десетичния знак.



Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

Друг начин за форматиране на низ са операторите {}, последвани от .format(*моята_променлива_1*, *моята_променлива_2*). Това е подобно на форматирането на f-string, което видяхме по-рано в този сборник. Операторите {} се използват като контейнери за променливата, но вместо да укажем името на променливата между къдравите скоби, оставяме скобите празни и подаваме променливата като аргумент във функцията .format(). Ако като аргументи се подадат няколко променливи, те трябва да бъдат разделени със запетаи. При изпълнението на програмата контейнерите ще бъдат заменени от реалните стойности на променливите по реда, по който са подадени променливите. Дробно число може да бъде допълнително форматирано с нотацията .xf, предшествана от двоеточие, като x указва броя на цифрите след десетичния знак. Вижте кода по-долу, който е идентичен с предишния пример, въпреки че използвахме този алтернативен метод.



Списъци

В Python списъкът (list) е тип данни - колекция от стойности, които са *подредени (поддържат индекси)* и *могат да се променят (променливи)*. Елементите в списъка се разделят със запетаи, а списъкът се записва в квадратни скоби [].

Празен списък се създава с празни квадратни скоби [] или с функцията list(). Списък с предварително дефинирани елементи се създава, като тези елементи се поставят в квадратни скоби, разделени със запетаи. Елементите могат да се дублират (не е задължително да са уникални).

```
empty_list = []
numbers = [42, 3.14, 1, 9]
```

Елементите/членовете на списък могат да бъдат от всякакъв тип данни, включително други списъци. В последния случай те се наричат вложени списъци.

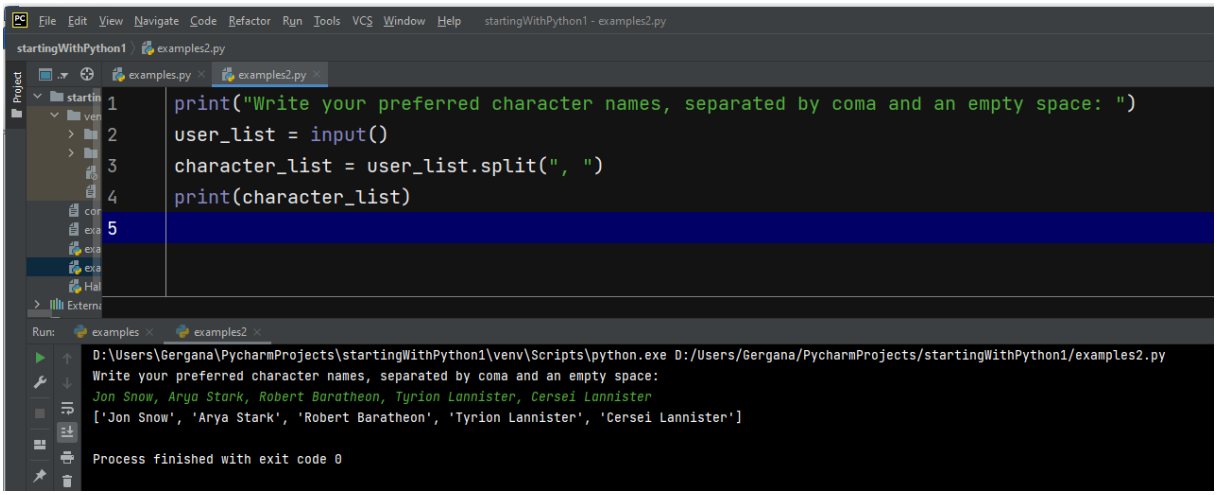
```
nested_numbers = [42, 3.14, 1, 9, [3.14, 0]]
```

Съвет:

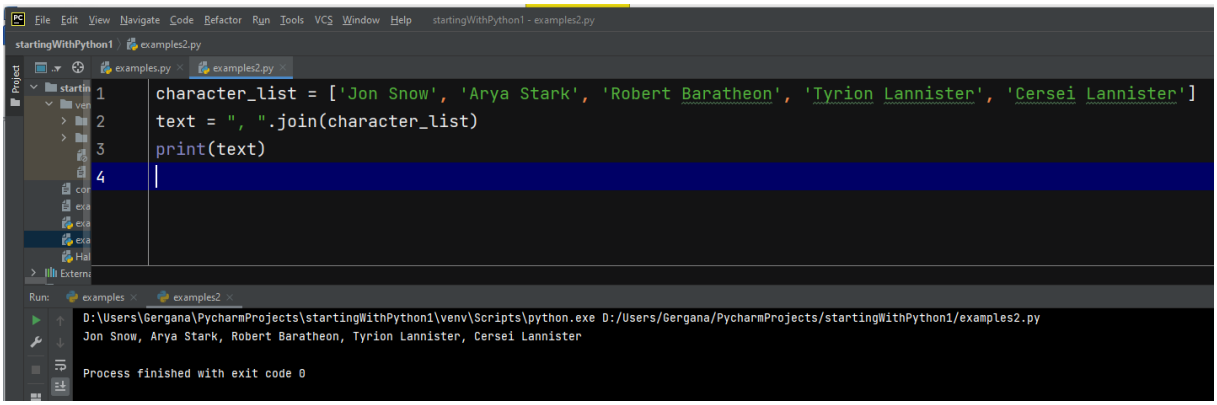
Полезен подход е да използваме *split*, за да разделим текст (низ) и директно да създадем списък от получените части. Това ще ни позволи да четем текст, въведен в конзолата или предоставен от потребителя, и да го съхраняваме в списък, както в примера по-долу. Разделянето се осъществява с функцията *потребителски_текст*.split("*разделител*"). Трябва да изберете разделител, който помага на Python да разпознае къде свършва една дума и започва друга. Всеки символ или група от символи може да бъде разделител.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



И обратното, можем да създадем текст от съществуващ списък с низове. Това става с командата *"разделител".*join*(име_на_списъка).* Не можем да обединяваме елементи от други типове данни, освен ако преди това не ги преобразуваме в низове.



Тъй като списъците са индексирани, можем да използваме индекси за достъп до член на списъка. Не забравяйте, че индексите започват от 0, а не от 1. Погледнете отново този списък

```
nested_numbers = [42, 3.14, 1, 9, [3.14, 0]]
```

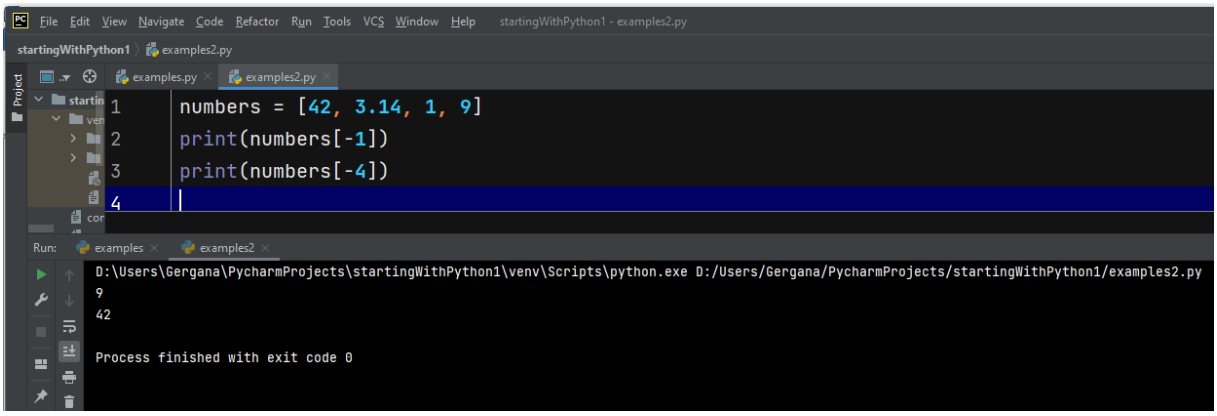
Елементът с индекс 1 в този списък е 3,14, а не 42. 42 е на индекс 0. По същия начин последният индекс е равен на *броя на елементите* - 1. Опитът за достъп до индекс, който е извън обхвата, е често срещана грешка в програмирането и ще доведе до прекратяване на програмата с грешка. Ако сме вложили списък в друг списък, можем да осъществим достъп до елементите на вложения списък с вложено индексиране (индексът, на който се намира вложения списък в основния списък е първи, а индексът на елемента във вложения списък е втори). Например, за да получим достъп до 0 в горния пример, трябва да напишем:

```
print(nested_numbers[4][1]) # Списъкът [3.14, 0] е петият елемент в списъка nested_numbers и се достъпва с [4]; 0 е вторият елемент във вложения списък и се достъпва с [1]
```

Подобно като при низовете, възможно е да се използват отрицателни индекси. Индекс –1 дава достъп до последния елемент, а – [*дължина на списъка*] дава достъп до първия елемент на списъка. Вижте примера по-долу.

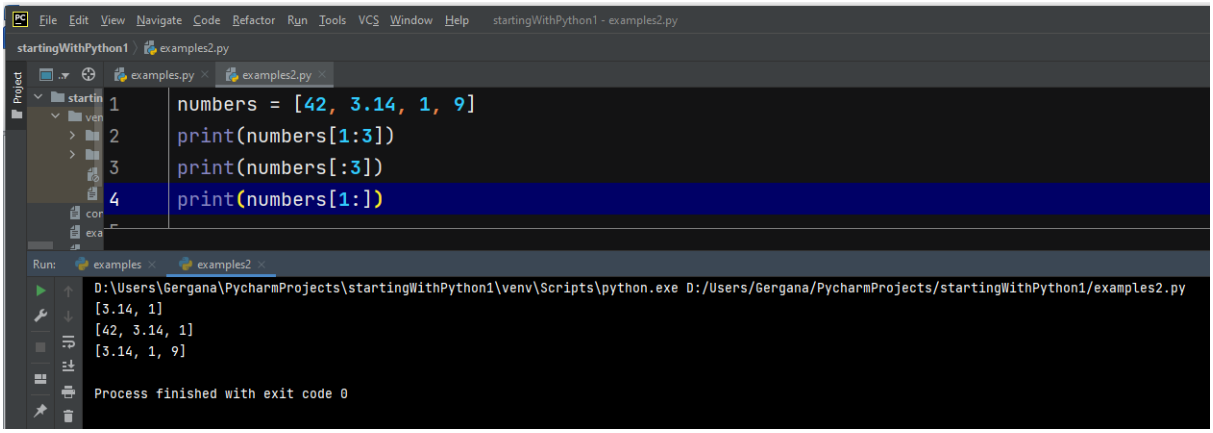
Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Можем да използваме отрез на списъци, за да върнем елементите в даден диапазон от начален до краен индекс. Това става, като посочим диапазона от индекси, който ни е необходим. Достъпва се началния индекс в диапазона, но крайния индекс не се достъпва (началният индекс е включващ, но крайният индекс е изключващ).

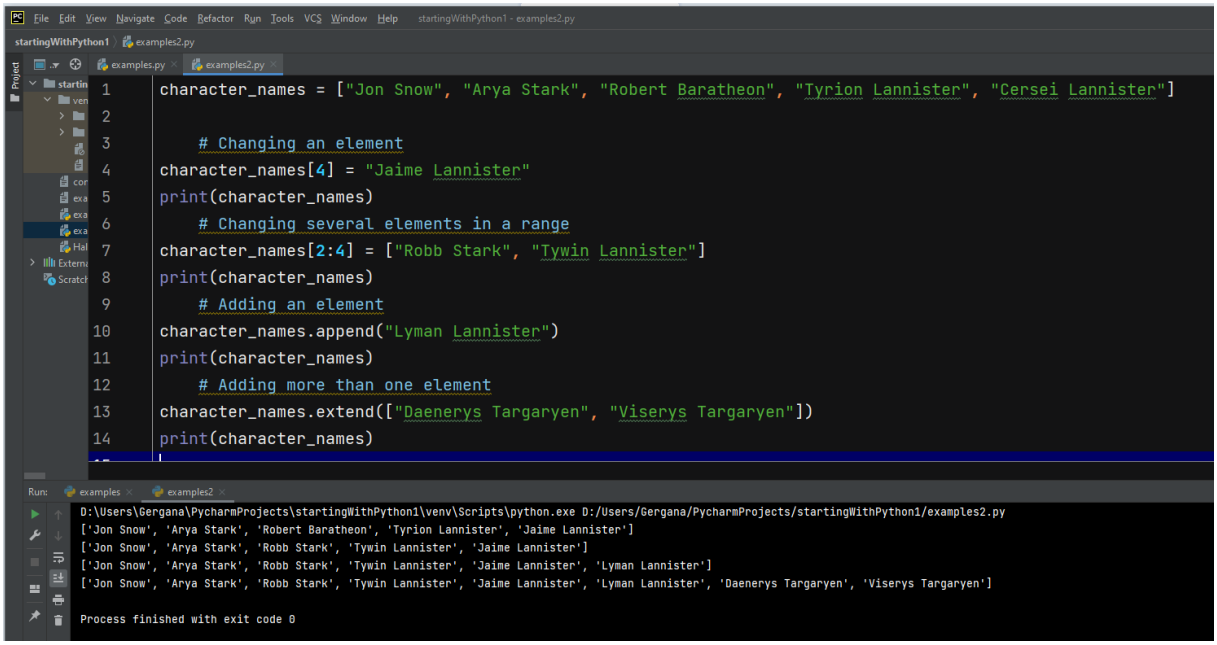
Командата *име_списък*[1:3] ще осъществи достъп и ще отпечата елементите на първия и втория индекс, но не и на третия. Ако не посочим начален индекс, а вместо това започнем от : (двоеточие), последвано от крайния индекс, ще получим достъп до всички елементи от началото на списъка до *крайния индекс* - 1. Ако не посочим краен индекс и започнем от начален индекс, последван само от : (двоеточие), ще получим достъп до всички елементи, започващи от началния индекс до края на списъка. Ако използваме [::] или [0:], ще получим достъп до всички елементи. Вижте примерите по-долу.



Можем да добавяме, премахваме и променяме елементи в списъка чрез достъп до техния индекс. Новите стойности се задават с помощта на оператора =. Нов елемент се добавя с метода *име_списък*.append(*елемент*). Новият елемент винаги се добавя в края на списъка. Можем да използваме метода *име_списък*.extend(*елемент_1, елемент_2*), за да добавим повече от един елемент, отново на опашката на списъка.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

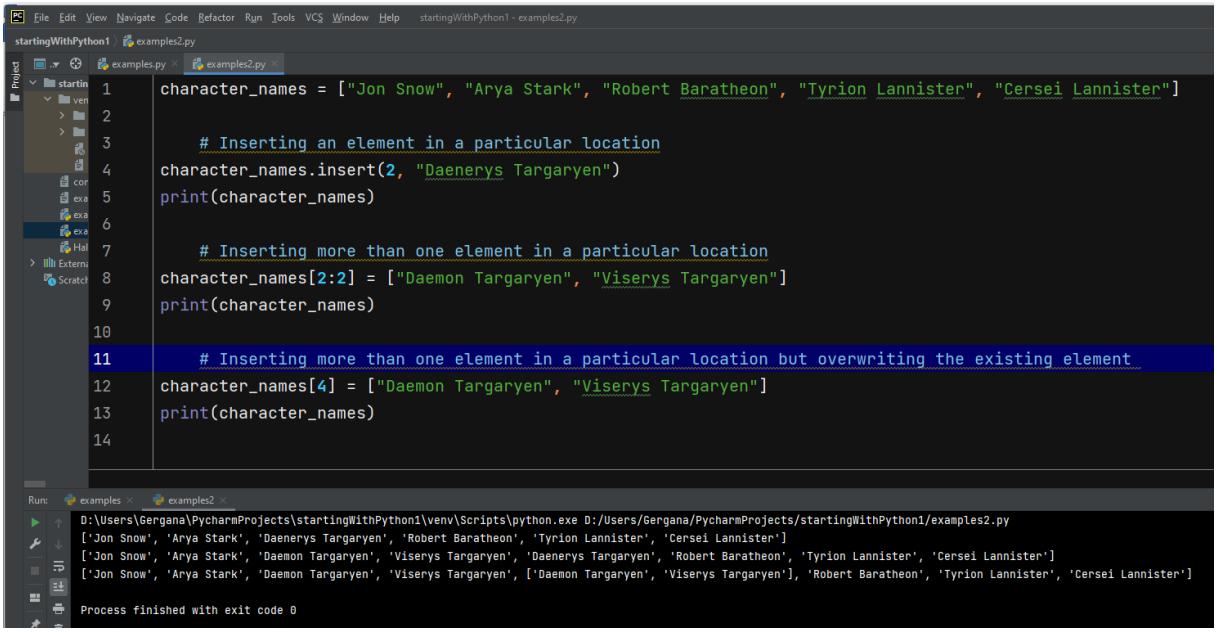
Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



```
character_names = ["Jon Snow", "Arya Stark", "Robert Baratheon", "Tyrion Lannister", "Cersei Lannister"]

    # Changing an element
character_names[4] = "Jaime Lannister"
print(character_names)
    # Changing several elements in a range
character_names[2:4] = ["Robb Stark", "Tywin Lannister"]
print(character_names)
    # Adding an element
character_names.append("Lyman Lannister")
print(character_names)
    # Adding more than one element
character_names.extend(["Daenerys Targaryen", "Viserys Targaryen"])
print(character_names)
```

```
D:\Users\Gergana\PycharmProjects\startingWithPython1\venv\Scripts\python.exe D:/Users/Gergana/PycharmProjects/startingWithPython1/examples2.py
['Jon Snow', 'Arya Stark', 'Robert Baratheon', 'Tyrion Lannister', 'Jaime Lannister']
['Jon Snow', 'Arya Stark', 'Robb Stark', 'Tywin Lannister', 'Jaime Lannister']
['Jon Snow', 'Arya Stark', 'Robb Stark', 'Tywin Lannister', 'Jaime Lannister', 'Lyman Lannister']
['Jon Snow', 'Arya Stark', 'Robb Stark', 'Tywin Lannister', 'Jaime Lannister', 'Lyman Lannister', 'Daenerys Targaryen', 'Viserys Targaryen']

Process finished with exit code 0
```

Ако трябва да добавим елемент с определен индекс, вместо това трябва да използваме метода *име_списък*.insert(*индекс*, *елемент*). Като аргументи се подават индексът, на който трябва да вмъкнем новия елемент, и самият нов елемент. Ако трябва да вмъкнем повече от един елемент на определено място, тогава присвояваме новите елементи (групирани в списък) на позиция [*индекс*:*индекс*]. Внимавайте да не ги присвоите на позиция [*индекс*], защото ще презапишете вече съществуващата стойност с новия списък.



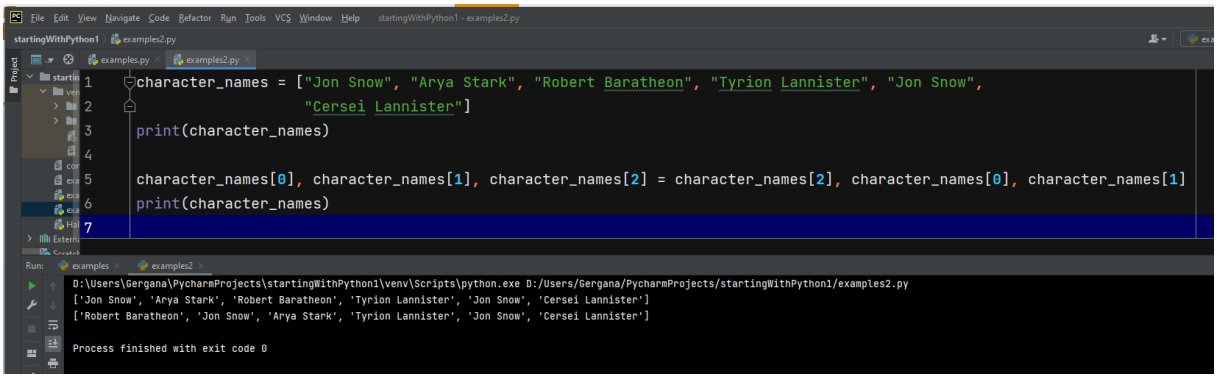
```
character_names = ["Jon Snow", "Arya Stark", "Robert Baratheon", "Tyrion Lannister", "Cersei Lannister"]

    # Inserting an element in a particular location
character_names.insert(2, "Daenerys Targaryen")
print(character_names)

    # Inserting more than one element in a particular location
character_names[2:2] = ["Daemon Targaryen", "Viserys Targaryen"]
print(character_names)

    # Inserting more than one element in a particular location but overwriting the existing element
character_names[4] = ["Daemon Targaryen", "Viserys Targaryen"]
print(character_names)
```

```
D:\Users\Gergana\PycharmProjects\startingWithPython1\venv\Scripts\python.exe D:/Users/Gergana/PycharmProjects/startingWithPython1/examples2.py
['Jon Snow', 'Arya Stark', 'Daenerys Targaryen', 'Robert Baratheon', 'Tyrion Lannister', 'Cersei Lannister']
['Jon Snow', 'Arya Stark', 'Daemon Targaryen', 'Viserys Targaryen', 'Daenerys Targaryen', 'Robert Baratheon', 'Tyrion Lannister', 'Cersei Lannister']
['Jon Snow', 'Arya Stark', 'Daemon Targaryen', 'Viserys Targaryen', ['Daemon Targaryen', 'Viserys Targaryen'], 'Robert Baratheon', 'Tyrion Lannister', 'Cersei Lannister']

Process finished with exit code 0
```

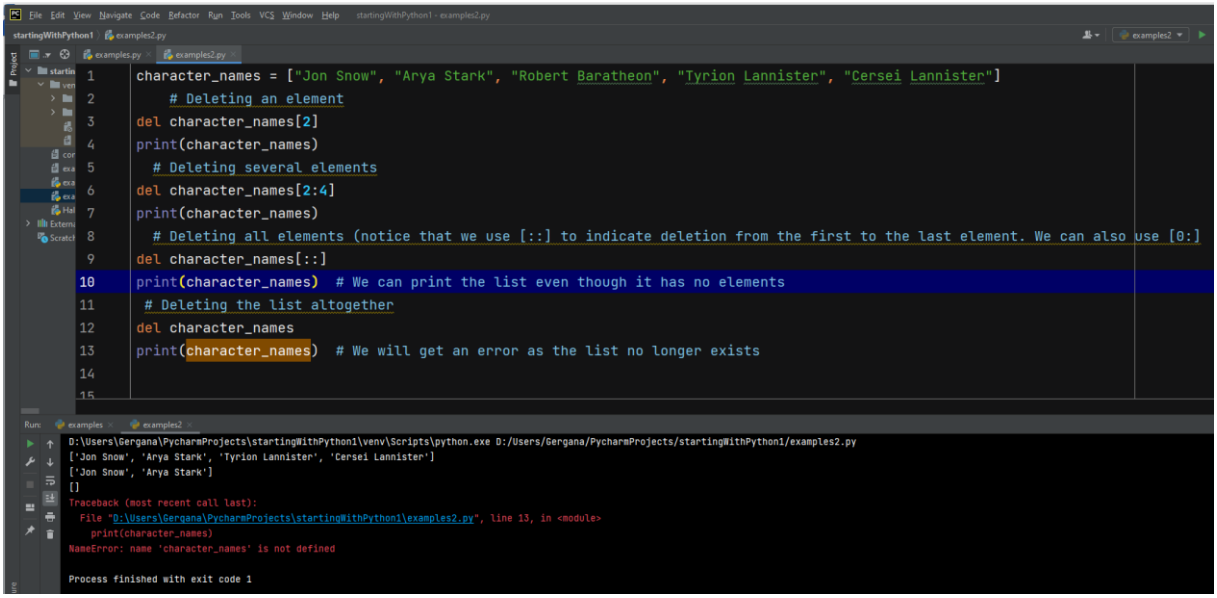
Възможно е също така да се разменят позициите (индексите) на елементите. Това става, като просто зададете на елементите различни индекси, както е показано в примера по-долу.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



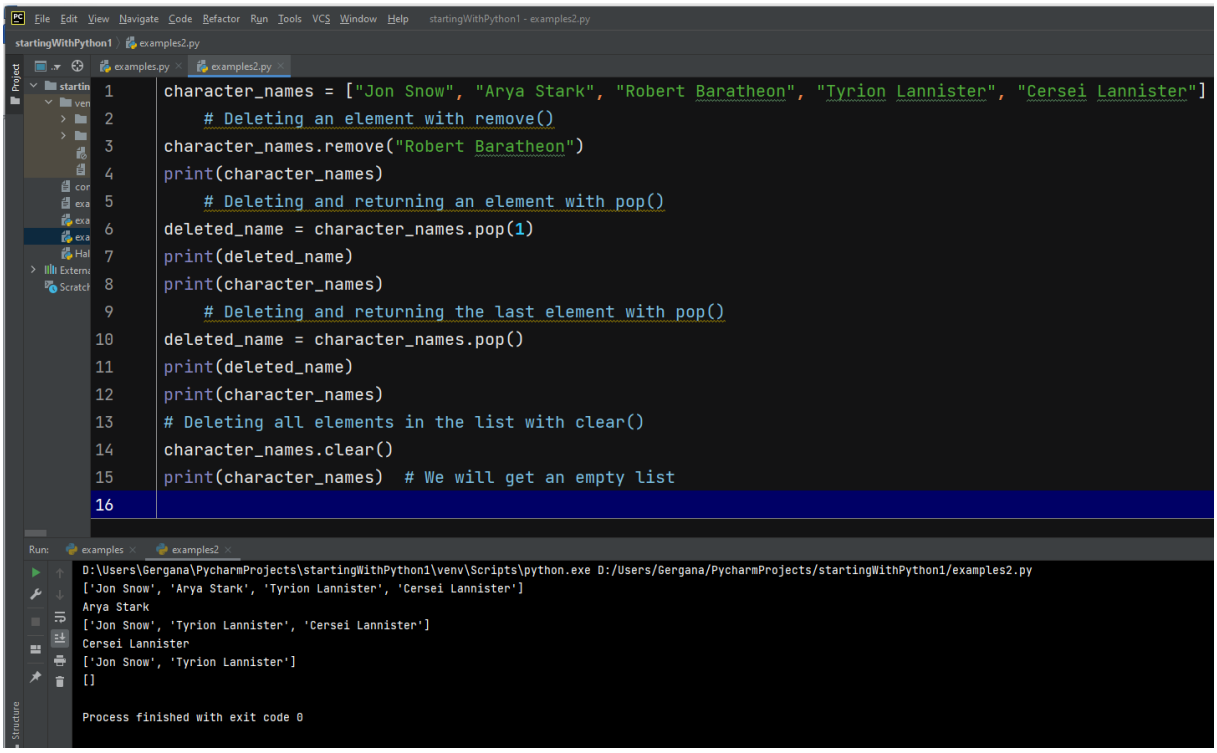
Изтриването на елемент се извършва чрез достъп до индекса на елемента и ключовата дума del. Можем да изтрием един или повече елементи, като посочим индекс или диапазон от индекси. Можем да изтрием и самия списък. Примери са показани по-долу.



Освен ключовата дума del можем да използваме метода *име_списък*.remove(*елемент*), като му подадем като аргумент елемента, който трябва да премахнем. Методът *име_списък*.pop(индекс) работи по същия начин, но трябва да подадем като аргумент индекса на елемента, който искаме да премахнем. Този метод също така връща елемента, който се премахва, така че можем да го използваме в програмата. Ако не подадем индекса като аргумент, pop() ще премахне последния елемент, което може да е полезно, ако трябва да използваме списъка като *стек (stack)*. Можем да изтрием всички елементи в списъка, като използваме метода *име_списък*.clear().

Съфинансиран от програма „Еразъм+“ на Европейския съюз

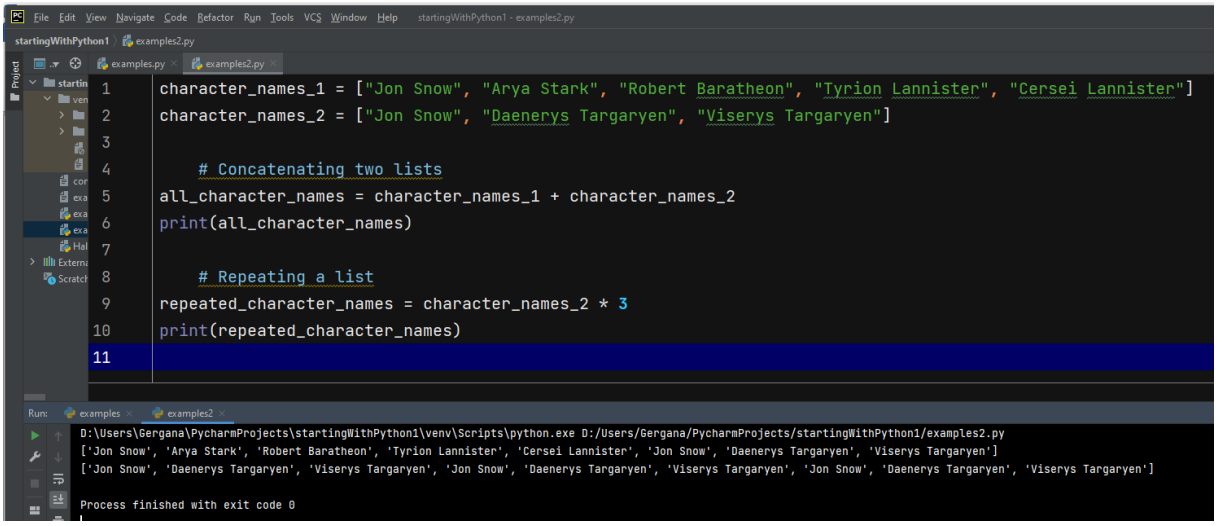
Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



```
character_names = ["Jon Snow", "Arya Stark", "Robert Baratheon", "Tyrion Lannister", "Cersei Lannister"]
    # Deleting an element with remove()
character_names.remove("Robert Baratheon")
print(character_names)
    # Deleting and returning an element with pop()
deleted_name = character_names.pop(1)
print(deleted_name)
print(character_names)
    # Deleting and returning the last element with pop()
deleted_name = character_names.pop()
print(deleted_name)
print(character_names)
# Deleting all elements in the list with clear()
character_names.clear()
print(character_names)  # We will get an empty list
```

```
D:\Users\Gergana\PycharmProjects\startingWithPython1\venv\Scripts\python.exe D:/Users/Gergana/PycharmProjects/startingWithPython1/examples2.py
['Jon Snow', 'Arya Stark', 'Tyrion Lannister', 'Cersei Lannister']
Arya Stark
['Jon Snow', 'Tyrion Lannister', 'Cersei Lannister']
Cersei Lannister
['Jon Snow', 'Tyrion Lannister']
[]

Process finished with exit code 0
```

Два списъка могат да бъдат комбинирани (конкатенирани) с оператора +. Един списък може да се повтори няколко пъти с помощта на оператора *, последван от броя на повторенията. Резултатът е нов списък.



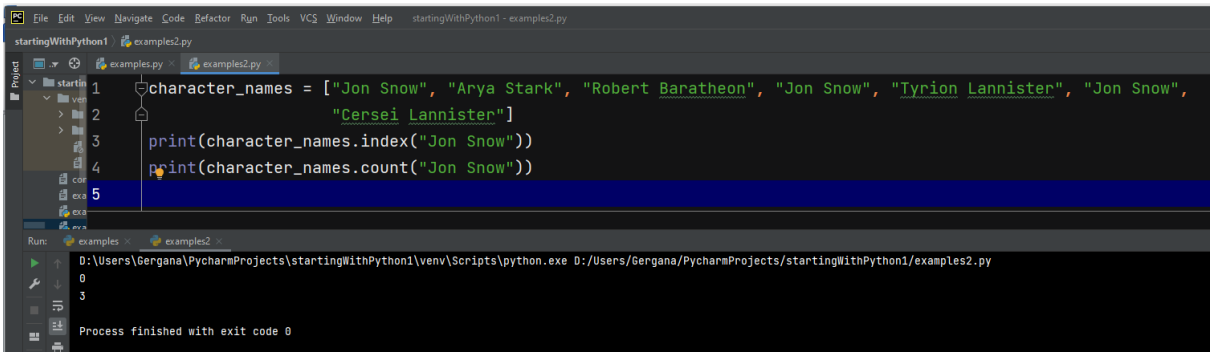
```
character_names_1 = ["Jon Snow", "Arya Stark", "Robert Baratheon", "Tyrion Lannister", "Cersei Lannister"]
character_names_2 = ["Jon Snow", "Daenerys Targaryen", "Viserys Targaryen"]

    # Concatenating two lists
all_character_names = character_names_1 + character_names_2
print(all_character_names)

    # Repeating a list
repeated_character_names = character_names_2 * 3
print(repeated_character_names)
```

```
D:\Users\Gergana\PycharmProjects\startingWithPython1\venv\Scripts\python.exe D:/Users/Gergana/PycharmProjects/startingWithPython1/examples2.py
['Jon Snow', 'Arya Stark', 'Robert Baratheon', 'Tyrion Lannister', 'Cersei Lannister', 'Jon Snow', 'Daenerys Targaryen', 'Viserys Targaryen']
['Jon Snow', 'Daenerys Targaryen', 'Viserys Targaryen', 'Jon Snow', 'Daenerys Targaryen', 'Viserys Targaryen', 'Jon Snow', 'Daenerys Targaryen', 'Viserys Targaryen']

Process finished with exit code 0
```

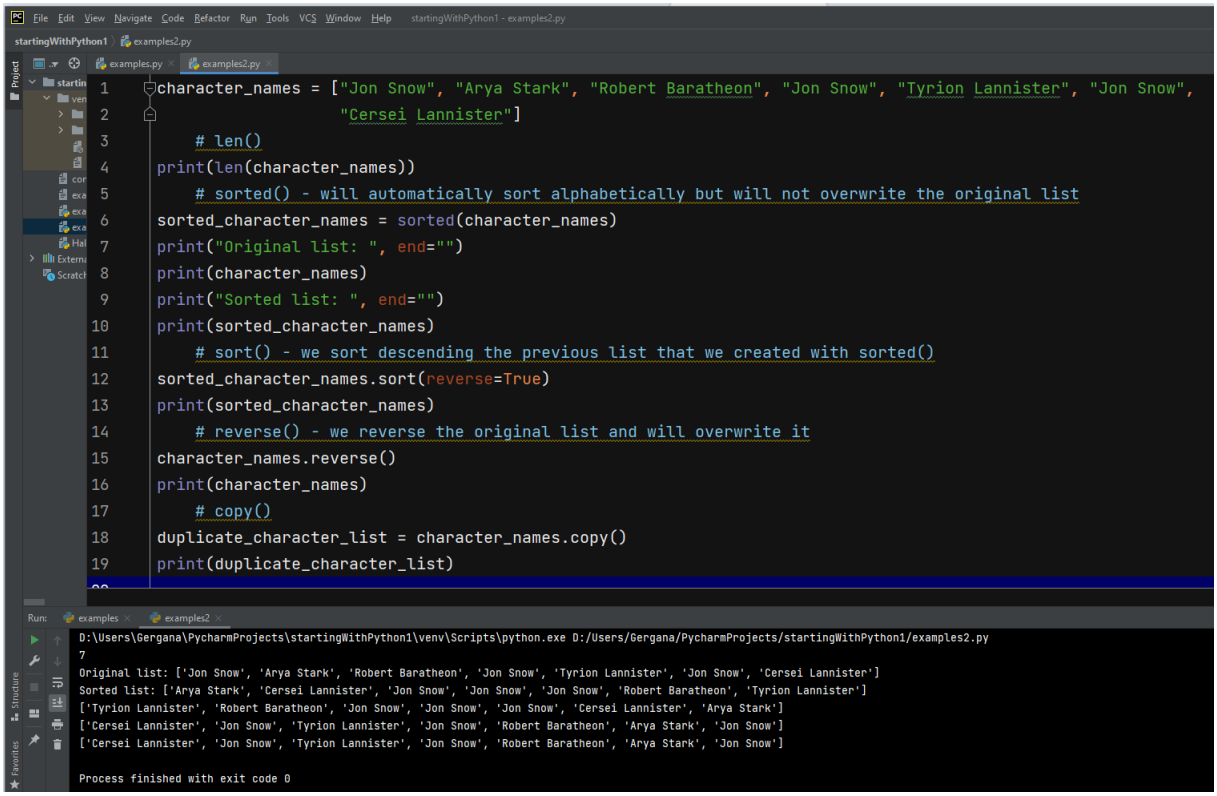
С метода *име_списък*.index(*елемент*) можем да получим индекса на елемент, за който знаем, че съществува в списъка. Въпреки това, ако елементът присъства в списъка няколко пъти, ще получим индекса само на първата поява на този елемент. От друга страна, можем да използваме метода *име_списък*.count(*елемент*), за да върнем броя на появяванията на определен елемент в списъка. В последния случай елементът се подава като аргумент.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



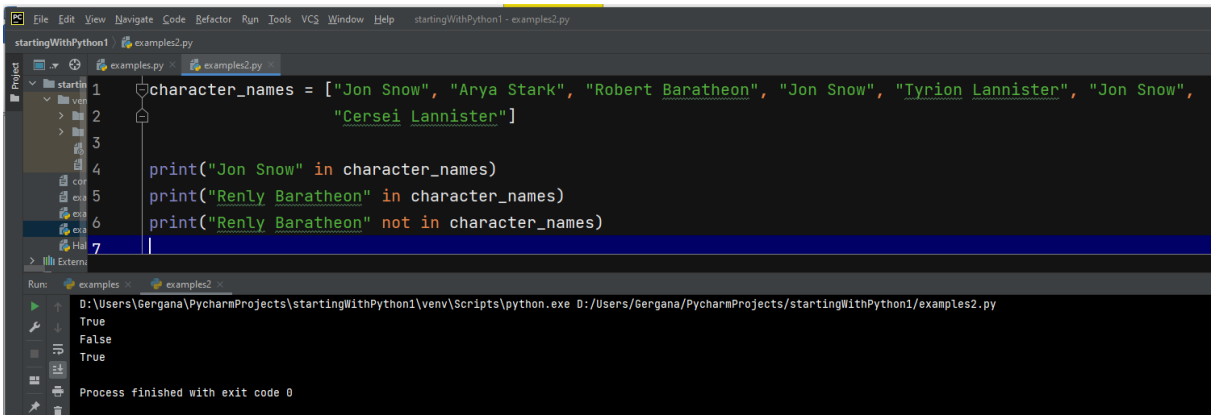
Други полезни вградени методи, които могат да ни помогнат да работим със списъци, са например len(*име_списък*) (връща броя на елементите в списъка), *име_списък*.sorted() (сортира списъка автоматично и връща резултата като нов списък; не променя оригиналния списък), *име_списък*.sort() (подрежда елементите в списъка автоматично или според параметър, който подаваме като аргумент, и презаписва първоначалния списък), *име_списък*.reverse() (обръща реда на елементите в списъка и презаписва първоначалния списък) и *име_списък*.copy() (връща нов списък, който е копие на първоначалния; не променя първоначалния списък). Вижте някои примери по-долу. Изпробвайте тези методи сами във вашата ИСР.



Често е полезно да се провери дали даден елемент присъства в списъка. Това се прави с помощта на ключовата дума in или ключовите думи not in. Програмата връща булева стойност - True или False.

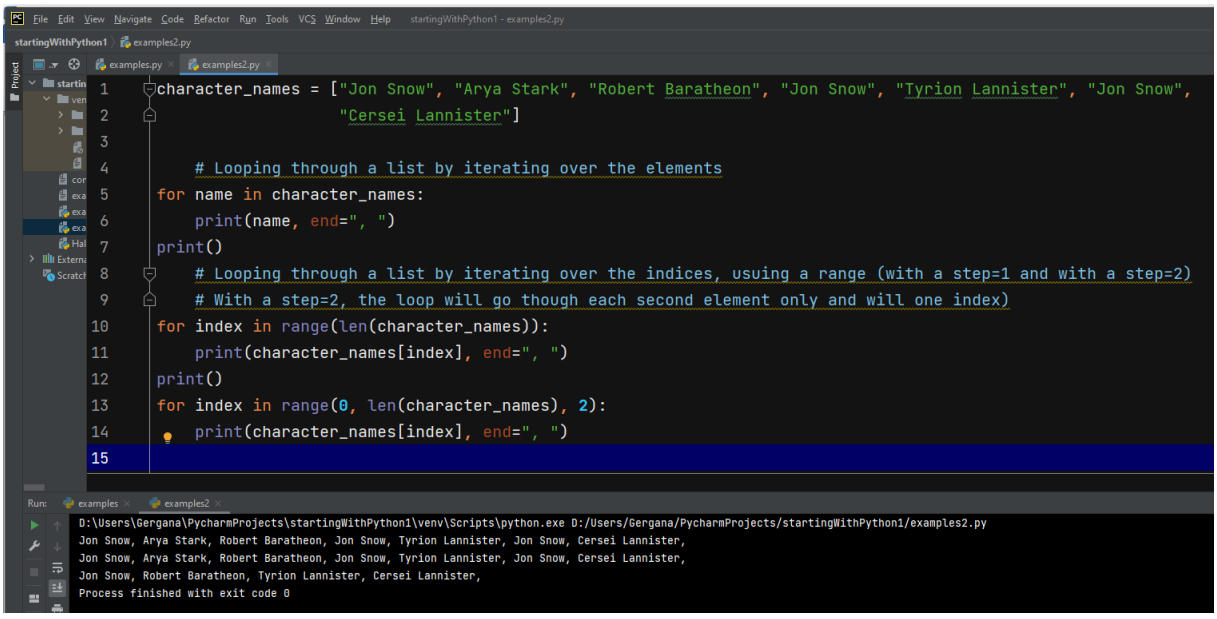
Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Нека сега научим как да итерираме в списък с помощта на цикъл. Можем да използваме *for* цикъл за итерация по елементите, както е показано в първия пример по-долу. Думата "*name*" е името на променливата, която съхранява стойността на текущия елемент при всяка итерация. На тази променлива се присвоява нова стойност по време на всяка итерация, докато преминаваме през целия списък. Можете да наречете тази променлива по подходящ начин в зависимост от това какво е включено във вашия списък (например число, температура и т.н.).

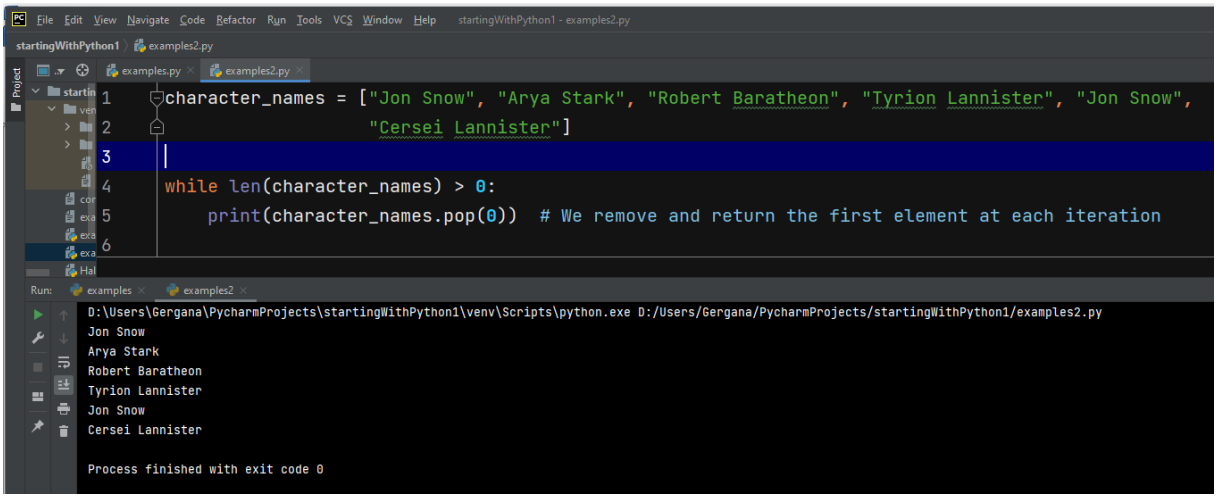
Можем също така да извършим итерация по индексите, като дефинираме диапазон. Допълнително предимство тук е, че можем да зададем стъпка за итерацията. Например, ако изберем стъпка=2, цикълът ще прескочи един индекс при преминаване към следващата итерация и ще премине само през всеки втори индекс. Синтаксисът тук е for *index* in range(стартов индекс, краен индекс, стъпка). Ако подадем само един аргумент, този аргумент ще се счита за краен индекс и цикълът ще итерира от началото на списъка до този индекс - 1. Не забравяйте, че крайният индекс не се включва в цикъла. Ако подадем само два аргумента, те ще се разглеждат като начален и краен индекс.



Като алтернатива можем да използваме *while* цикъл за итерация по списък. Това обаче изисква да използваме функцията pop(), за да премахнем всеки елемент, над който вече сме преминали в цикъла. Така в *while* цикъла можем да извършваме итерации, докато списъкът не е празен. Когато преминем през последния елемент, ще го премахнем с функцията pop() и списъкът ще остане празен, което ще сложи край на цикъла. За да проверим дали списъкът не е празен, използваме функцията len().

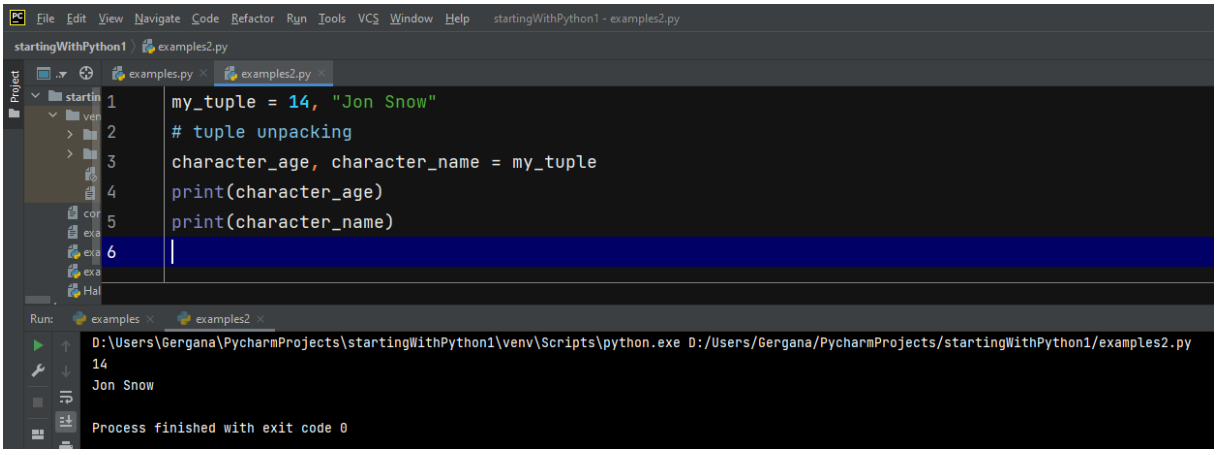
Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Кортежи

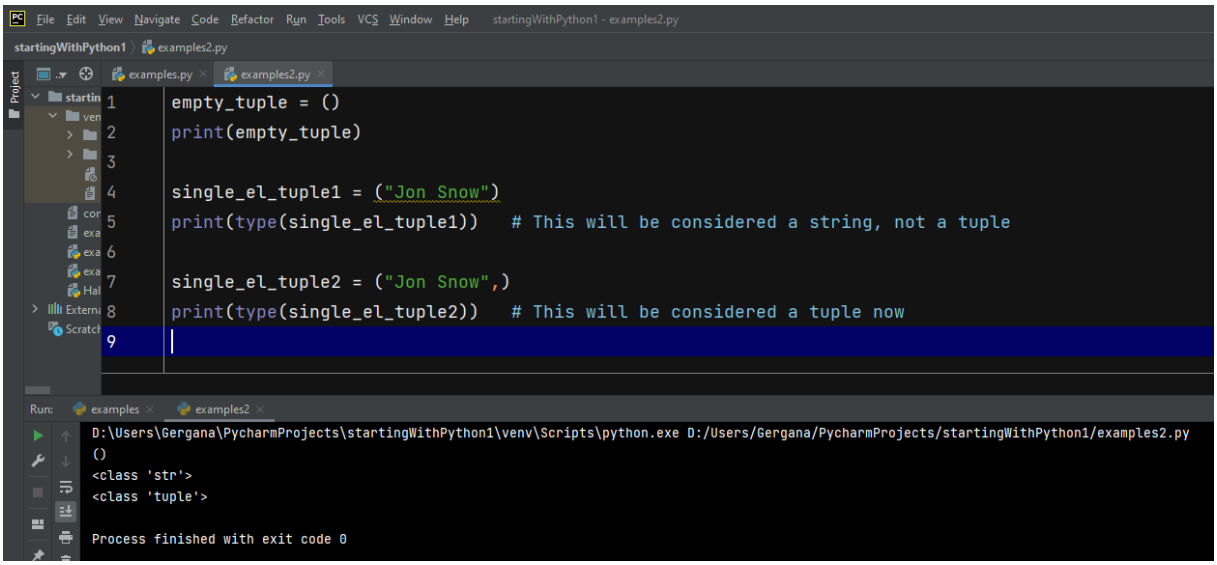
В Python кортежът (tuple) е тип данни - колекция от стойности, които са *подредени* и *непроменяеми*. Той е много подобен на списък, като единствената разлика е, че елементите не могат да бъдат променяни, след като са поставени в кортежа. Също така не можем да добавяме или премахваме елементи от кортежа. Елементите в кортежа са разделени със запетаи, а кортежът е записан в кръгли скоби (). Елементите могат да се повтарят. Понякога можете да видите кортежи, създадени без скоби, но този метод не се препоръчва да се използва. Елементите вътре в кортежа могат да бъдат от всякакъв тип данни (напр. цели числа, дробни числа, низове и дори колекции, като други кортежи или списъци), като можем да създадем и кортеж с различни типове данни в него. Възможно е да се извърши "разопаковане" (unpacking) на кортежа, като стойностите, които се съдържат в кортежа, се присвояват на променливи. Вижте пример за "разопаковане" на кортеж.



За да инициализираме празен кортеж, му присвояваме (). За да инициализираме кортеж с един елемент, не е достатъчно да поставим този елемент в кръгли скоби, защото Python ще приеме, че типа данни е този на самия елемент. Трябва да добавим запетая след този единствен елемент, за да укажем, че имаме кортеж с един елемент.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

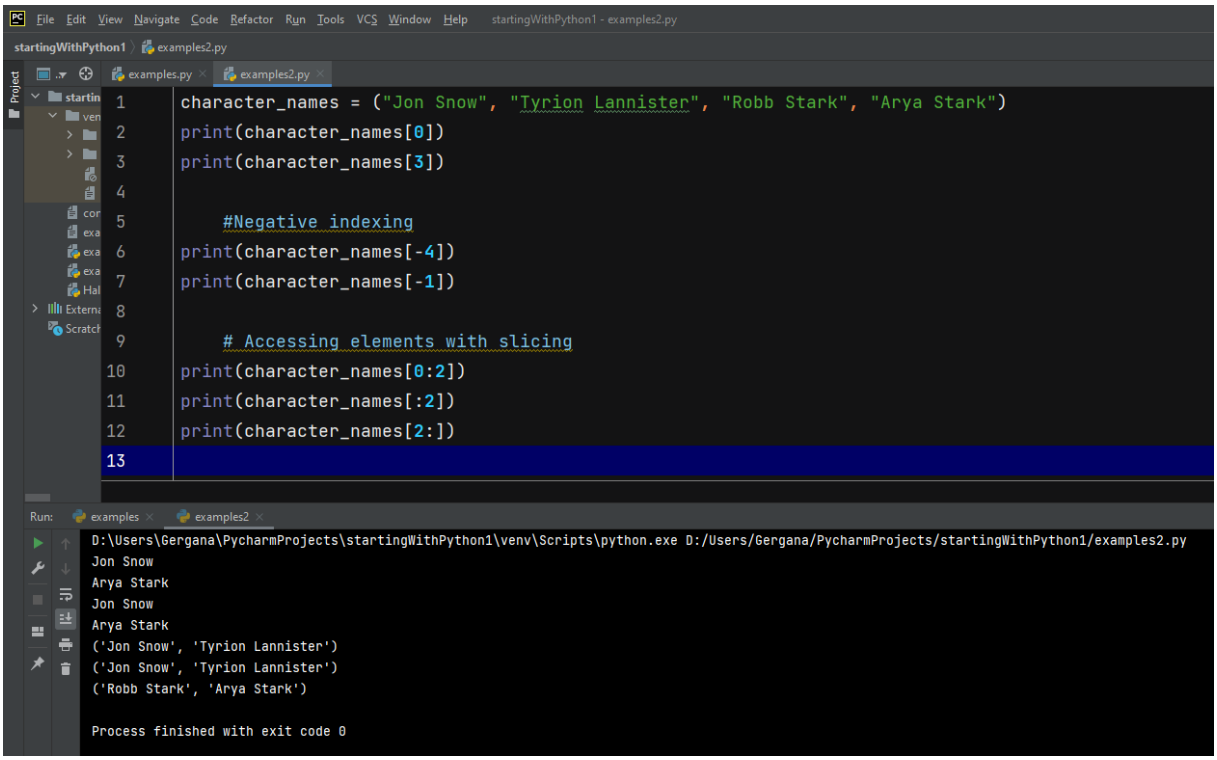
Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



Елементите на кортежа са подредени и са индексирани. Следователно достъпът до всеки елемент се осъществява чрез неговия индекс. Точно както при низовете и списъците, индексите започват от 0, а последният индекс е дължината на кортежа – 1. Извикването на индекс, който е равен на дължината на кортежа, ще доведе до грешка "Index out of range error". Позволено е отрицателно индексиране: последният индекс е -1, а първият индекс е -(*дължина на кортежа*), т.е. в случая на кортеж с 6 елемента с име *моят_кортеж* първият индекс може да се извика с *моят_кортеж*[-6], а последният индекс - с *моят_кортеж*[-1]. Достъпът до повече от един елемент може да бъде осъществен чрез *отрез (slicing),* като в квадратните скоби се поставя обхватът на отреза, например ако ни е необходим отрез от индекс 1 до индекс 5, се записва [1:6]. Както при низовете и списъците, началният индекс се достъпва, но крайният не, така че в нашия случай последният достъпен индекс ще бъде 5. Ако в квадратните скоби посочим само един индекс, последван от двоеточие, например [2:], тогава ще получим достъп до всички индекси от посочения индекс (включително и него) до края на кортежа. По същия начин, ако посочим един индекс, предшестван от двоеточие, ще получим достъп до всички индекси от началото на кортежа до посочения индекс (без този индекс). Вижте всичко това нагледно в примера.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

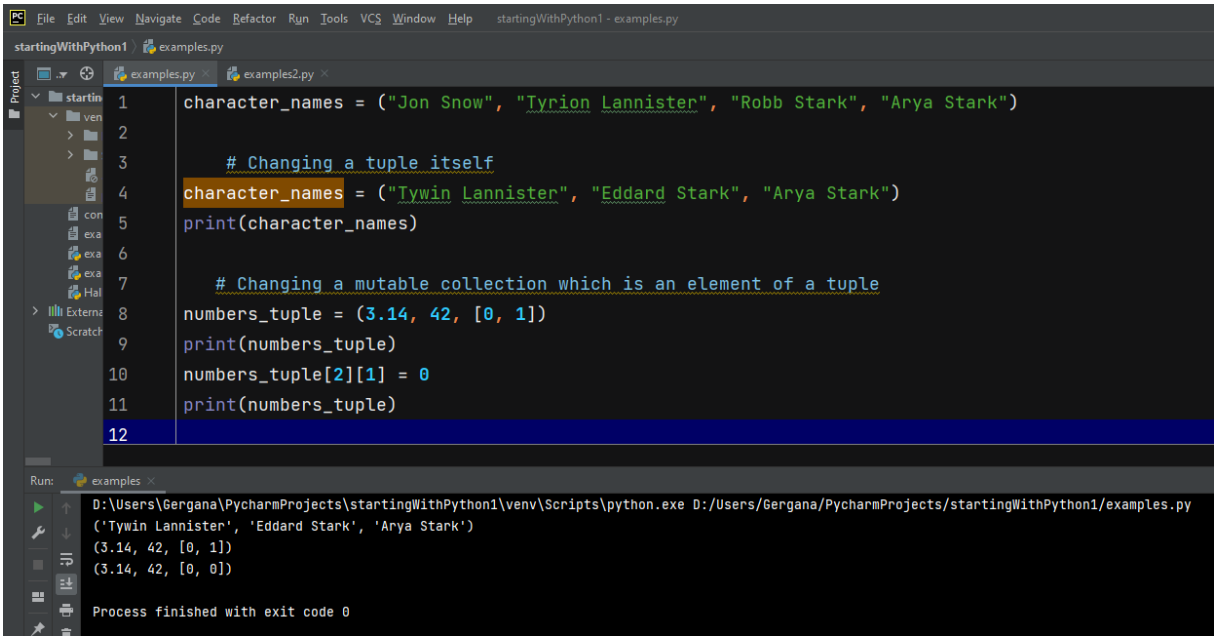


```
character_names = ("Jon Snow", "Tyrion Lannister", "Robb Stark", "Arya Stark")
print(character_names[0])
print(character_names[3])

    #Negative indexing
print(character_names[-4])
print(character_names[-1])

    # Accessing elements with slicing
print(character_names[0:2])
print(character_names[:2])
print(character_names[2:])
```

```
D:\Users\Gergana\PycharmProjects\startingWithPython1\venv\Scripts\python.exe D:/Users/Gergana/PycharmProjects/startingWithPython1/examples2.py
Jon Snow
Arya Stark
Jon Snow
Arya Stark
('Jon Snow', 'Tyrion Lannister')
('Jon Snow', 'Tyrion Lannister')
('Robb Stark', 'Arya Stark')

Process finished with exit code 0
```

За разлика от елементите на списък, елементите на кортеж не могат да се променят. Единственото изключение от това правило е, когато елементът е променлива колекция, например списък. За да променим такъв елемент го достъпваме с две квадратни скоби [][], съдържащи съответно индекса на променливата колекция (например списък) и индекса на елемента в тази променлива колекция, който искаме да променим. След това присвояваме на този елемент нова стойност.

Можем да променим самия кортеж, ако му присвоим различни стойности. Разгледайте примерите по-долу и опитайте сами в PyCharm.



```
character_names = ("Jon Snow", "Tyrion Lannister", "Robb Stark", "Arya Stark")

    # Changing a tuple itself
character_names = ("Tywin Lannister", "Eddard Stark", "Arya Stark")
print(character_names)

   # Changing a mutable collection which is an element of a tuple
numbers_tuple = (3.14, 42, [0, 1])
print(numbers_tuple)
numbers_tuple[2][1] = 0
print(numbers_tuple)
```

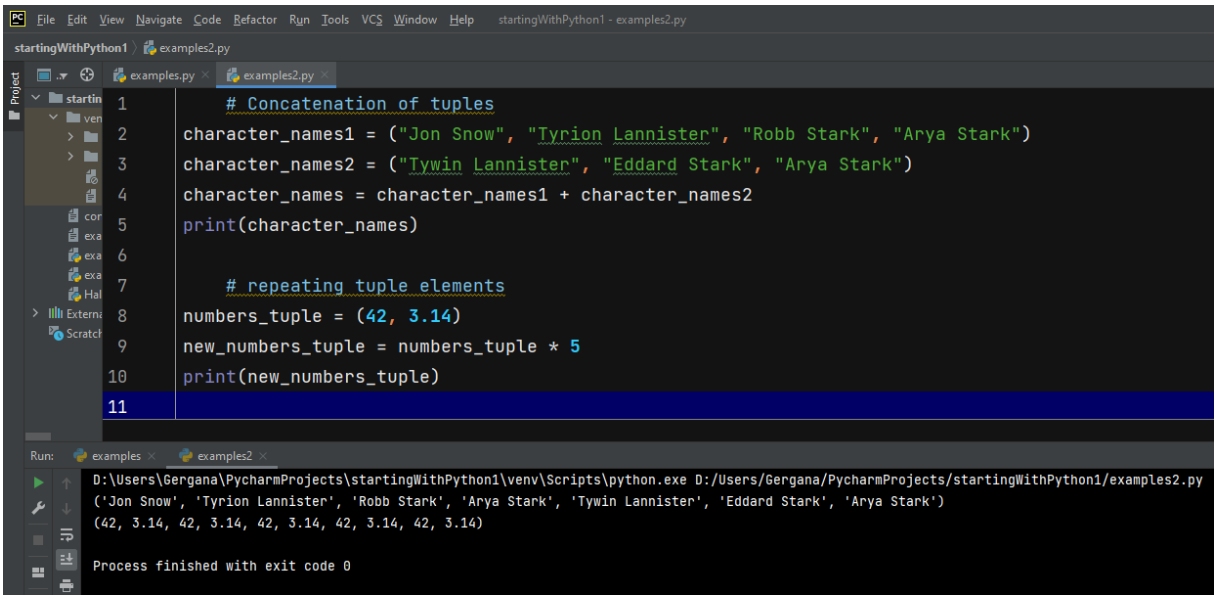
```
D:\Users\Gergana\PycharmProjects\startingWithPython1\venv\Scripts\python.exe D:/Users/Gergana/PycharmProjects/startingWithPython1/examples.py
('Tywin Lannister', 'Eddard Stark', 'Arya Stark')
(3.14, 42, [0, 1])
(3.14, 42, [0, 0])

Process finished with exit code 0
```

Можем да комбинираме няколко кортежа заедно с оператора +, като действието се нарича *конкатенация*. Ако конкатенираните кортежи съдържат някои идентични елементи, те ще бъдат дублирани. Можем също така да възпроизведем няколко пъти всички елементи на

Съфинансиран от програма „Еразъм+“ на Европейския съюз

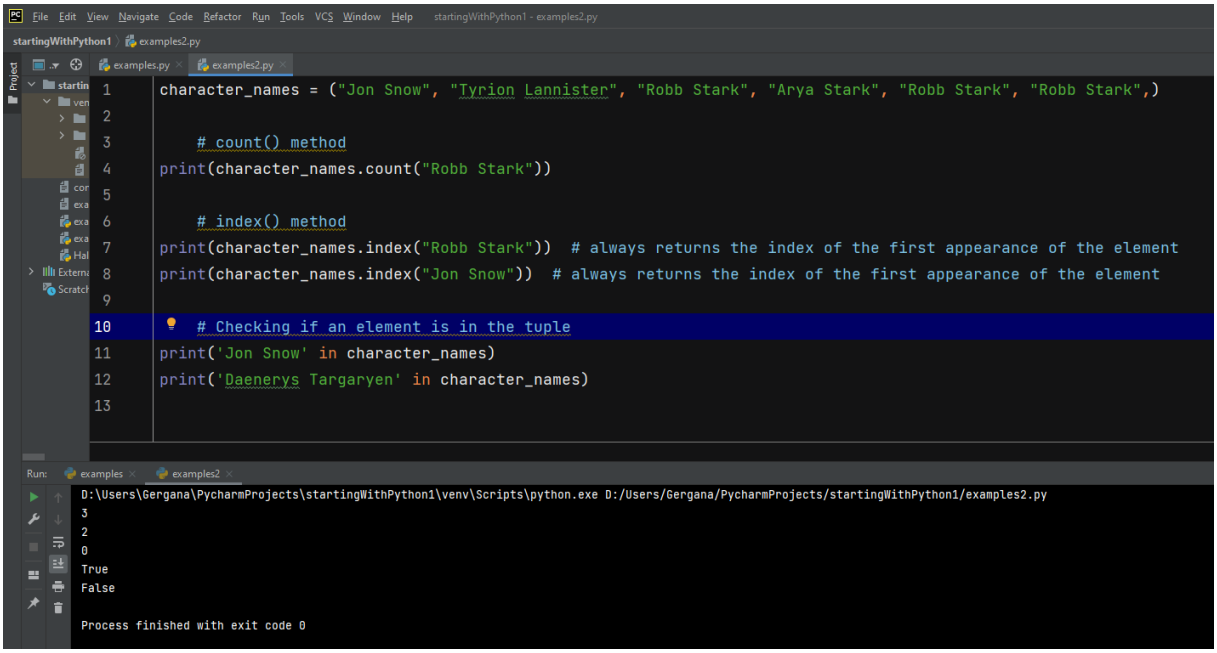
Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

един кортеж, като използваме оператора *, последван от число, указващо колко пъти ще бъдат повторени елементите. И двете операции създават нов кортеж. Вижте примерите по-долу.



Не можем да изтриваме или добавяме елементи в кортежа. Можем да изтрием само целия кортеж с командата del *моят_кортеж*. Опитайте сами в PyCharm.

В Python има няколко метода, които ни помагат да работим с кортежи. Първият от тях е методът count(*елемент*). В него като аргумент се подава елемент от кортежа и методът връща броя на появяванията на посочения елемент в кортежа. Методът index(*елемент*) приема като аргумент елемент от кортежа и връща индекса на първата поява на този елемент. Съществува и метод за проверка дали даден елемент се намира в кортежа или не. Методът връща True или False. Синтаксисът е *моят_елемент* in *моят_кортеж* или *моят_елемент* not in *моят_кортеж*. По същество тези методи са идентични с методите, използвани в списъците. Вижте примерите по-долу и ги изпробвайте сами.

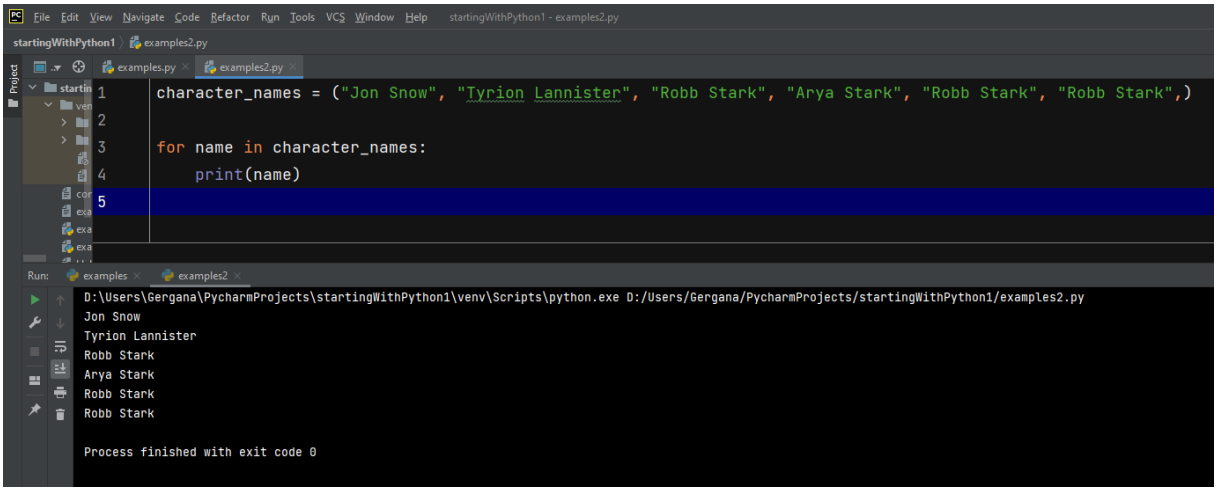


И накрая, нека научим как да итерираме по кортежа. Подходът е същия като при списъците. Запомнете, че синтаксисът на *for* цикъла е следният: ключова дума for,

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

последвана от име на променливата, представляваща всеки текущ елемент, последвана от ключовата дума in и последвана от името на променливата на кортежа (или самия кортеж, изписан в кръгли скоби). Сега, вътре във *for* цикъла можем да напишем код, използващ текущия елемент, извлечен по време на текущата итерация. Вижте примера.



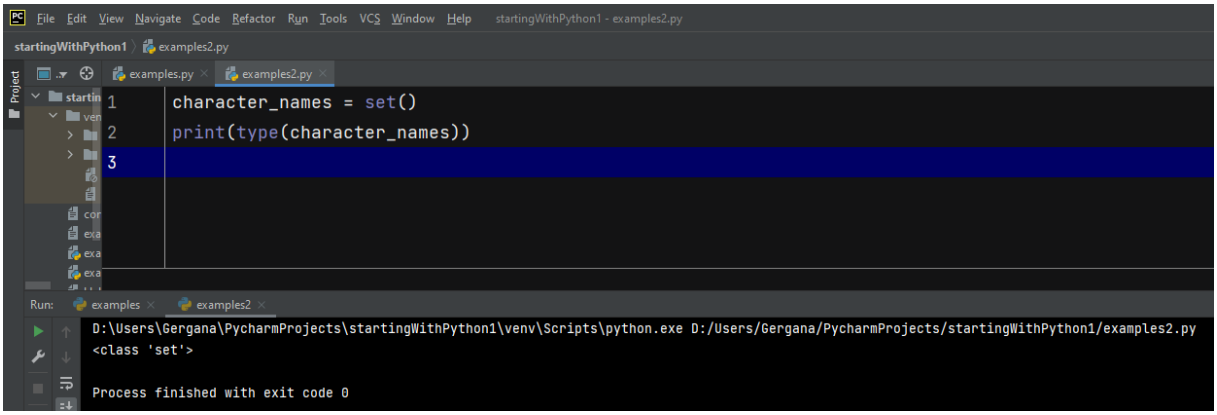
Съвет за това защо бихме използваме кортеж:

Корпусите съдържат непроменящи се елементи и могат да се използват за получаване на ключове за речник. Списъците не могат да се използват по този начин.

Сетове (sets)

В Python set е тип данни - колекция от елементи, които *не са подредени* и са *неизменни*. Самият сет обаче е променлив, така че можем да добавяме или премахваме елементи от него. Тъй като елементите не са подредени, не можем да използваме индекса на даден елемент, за да го извлечем. Елементите в сета са разделени със запетаи, а кортежът е записан в къдрави скоби {}. Елементите в един сет могат да бъдат от всякакъв тип данни (например цели числа, низове) и е възможно да се създаде сет с различни типове данни в него. Елементите в сета са уникални и ако се опитаме да добавим дубликати, те няма да бъдат взети под внимание.

Ако трябва да инициализираме празен сет, не можем да го направим, като присвоим празни къдрави скоби, защото това ще създаде празен речник. Можем да създадем празен сет с функцията set(), без да подаваме никакъв аргумент в нея.



Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

Можем да добавяме елементи към сета, като използваме метода add(*елемент*) и му подадем новия елемент като аргумент. Ако трябва да добавим няколко елемента към сета, използваме метода update([*елемент_1, елемент_2*]) и добавяме тези елементи като аргументи, разделени със запетаи и затворени в квадратни скоби. В примерите по-долу ще забележите, че отпечатания сет има различен ред от реда, в който добавихме елементите. Всъщност всяко ново изпълнение на командата print() може да отпечатва елементите в различен ред.

Можем да премахваме елементи от сета с методите discard(*елемент*) и remove(*елемент*). И двата метода ще премахнат от сета елемента, който сме подали като аргумент. Ако обаче елементът, който се опитваме да премахнем, не съществува в сета, discard() няма да направи нищо, докато remove() ще генерира грешка, за да предупреди за опит за премахване на несъществуващ елемент. Изборът между двете възможности ще зависи от това дали искаме програмата да се срине и да посочи грешка, или просто да се опита да премахне елемента и да продължи без прекъсване, ако премахването е неуспешно. Методът pop()премахва елемент от сета и го връща на програмата, но го прави с произволен елемент, тъй като сетът не е подреден. Обърнете внимание, че методът pop() връща стойността, която изтрива, така че можете да използвате тази стойност във вашата програма. Примерите по-долу илюстрират всички функции.

```
character_names = {"Jon Snow", "Tyrion Lannister", "Robb Stark"}
print(character_names)
    # add() function
character_names.add("Jaime Lannister")
print(character_names)
    # update() function
character_names.update(["Daenerys Targaryen", "Tywin Lannister"])
print(character_names)
    # remove() function with existing element
character_names.remove("Daenerys Targaryen")
print(character_names)
    # update() function with non-existing element
character_names.discard("Arya Stark")
print(character_names)
    # pop() function with saving he removed value into a variable
removed_name = character_names.pop()
print(removed_name)
    #remove() function  with non-existing element will generate an error
character_names.remove("Arya Stark")
print(character_names)
```

```
D:\Users\Gergana\PycharmProjects\startingWithPython1\venv\Scripts\python.exe D:/Users/Gergana/PycharmProjects/startingWithPython1/examples2.py
Traceback (most recent call last):
  File "D:\Users\Gergana\PycharmProjects\startingWithPython1\examples2.py", line 19, in <module>
    character_names.remove("Arya Stark")
KeyError: 'Arya Stark'
{'Tyrion Lannister', 'Jon Snow', 'Robb Stark'}
{'Jaime Lannister', 'Tyrion Lannister', 'Jon Snow', 'Robb Stark'}
{'Jaime Lannister', 'Tyrion Lannister', 'Daenerys Targaryen', 'Robb Stark', 'Jon Snow', 'Tywin Lannister'}
{'Jaime Lannister', 'Tyrion Lannister', 'Robb Stark', 'Jon Snow', 'Tywin Lannister'}
{'Jaime Lannister', 'Tyrion Lannister', 'Robb Stark', 'Jon Snow', 'Tywin Lannister'}
Jaime Lannister
```

Можем също така да премахнем всички елементи от даден сет, като използваме метода clear(). Копие на сета може да се създаде с метода copy().

Ако имаме няколко сета, можем да ги манипулираме чрез математически операции - *обединение*, *пресичане*, *разлика* и *симетрична разлика*.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

Обединението на два сета създава нов сет, който съдържа всички елементи от двата сета. Въпреки това дублиращите се елементи ще бъдат премахнати. *Обединяването* се извършва с помощта на оператора | или метода union().

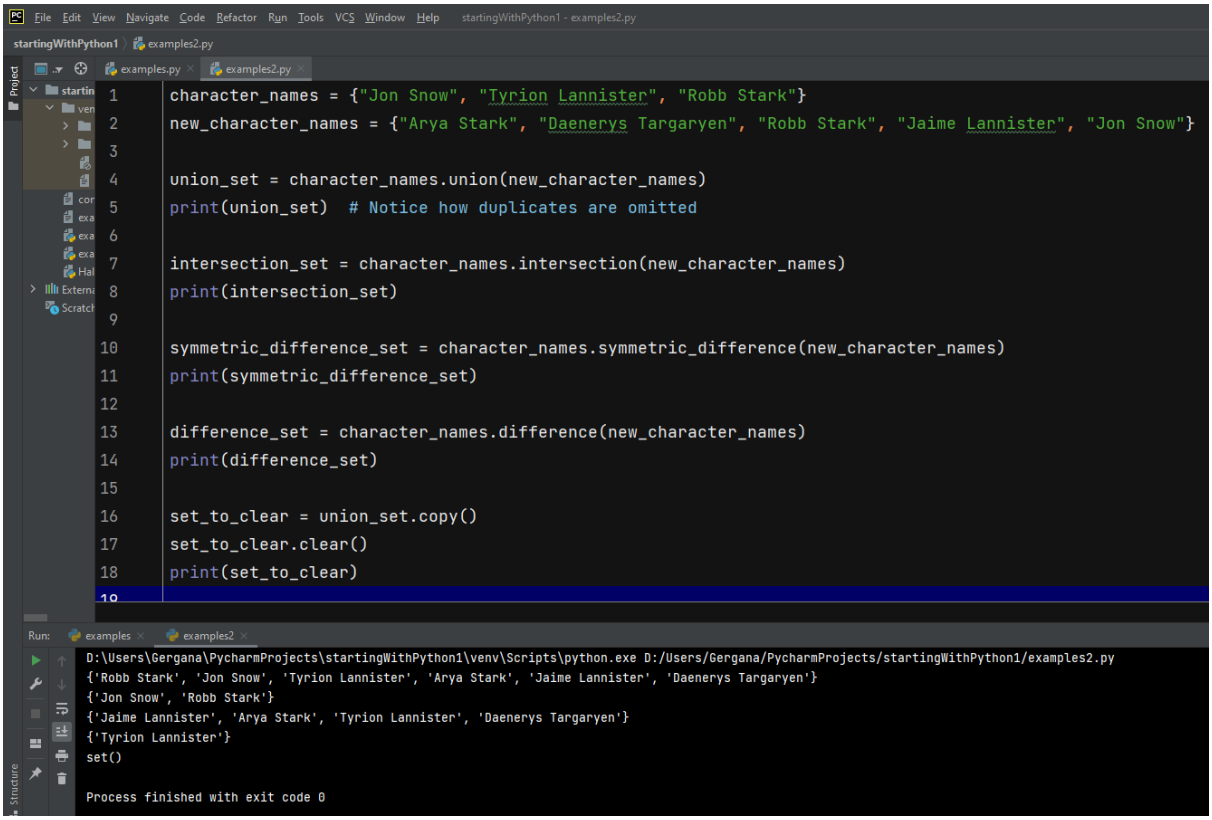
Пресичането на два сета създава нов сет, съдържащ само елементите, които са общи за двата първоначални сета. *Пресичането* се извършва с помощта на оператора & или метода intersection().

Разликата между два сета създава нов сет, съдържащ само елементите, които присъстват в първия сет, но не присъстват във втория. *Разликата* се извършва с помощта на оператора – или метода difference(). За разлика от другите две операции, тук трябва да внимаваме от кой сет изваждаме и кой сет се изважда - напр. резултатът от сет 1 – сет 2 ще бъде много по-различен от резултата от сет 2 – сет 1.

Симетричната разлика на два сета създава нов сет, съдържащ само елементите, които са в първия или втория сет, но не и в двата (т.е. общите елементи ще бъдат изключени от новия сет, така че това действие е обратното на *пресичането*). *Симетричната разлика* се извършва с помощта на оператора ^ или метода symmetric_difference().

Същите методи могат да се използват с добавена към тях функция _update() - intersection_update(), difference_update(), symmetric_difference_update (), union_update() - в този случай първото множество ще бъде заменено с резултата от съответната операция.

По-долу са показани методите за *обединение*, *пресичане*, *разлика* и *симетрична разлика*, както и методите copy() и clear(). Можете да тествате всичко останало сами.



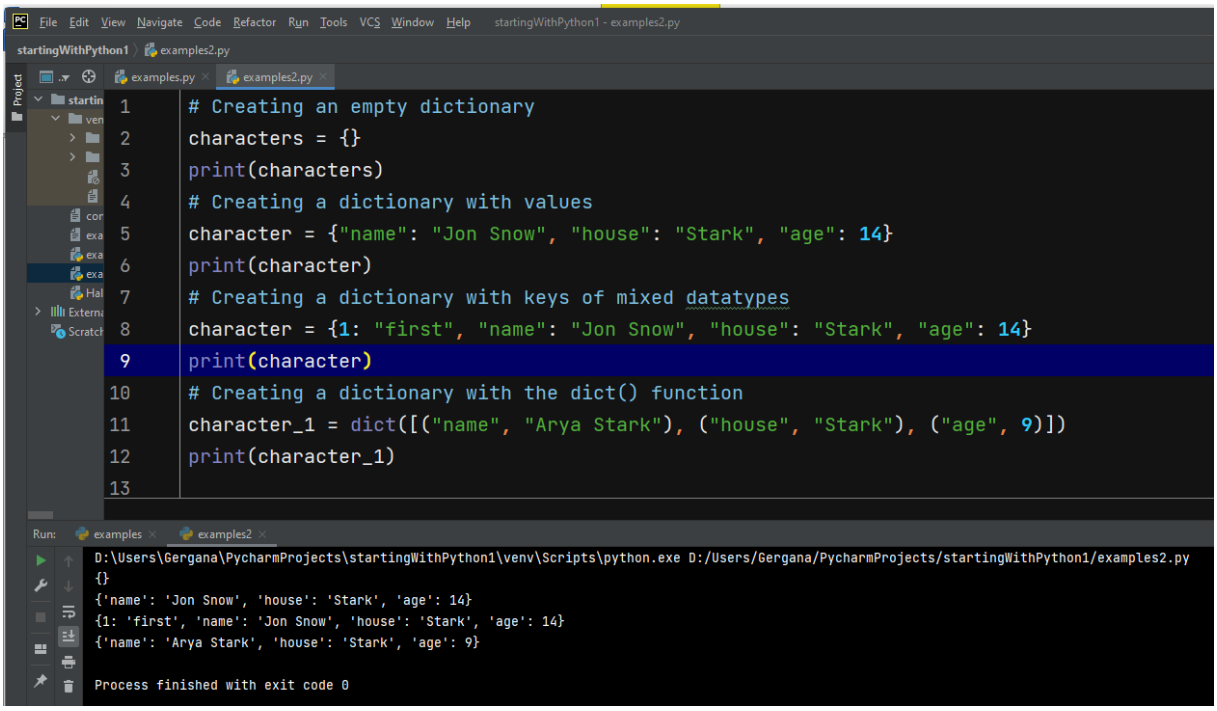
Речници

В Python речникът е тип данни - колекция от стойности, които са *подредени* (след Python 3.7) и *променливи*. Състои се от двойки ключ и стойност, свързани с ключа (*ключ: стойност*).

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

Двойките ключ-стойност в речника се разделят със запетаи, а речникът се записва във къдрави скоби {}. Можем да създадем речник и с функцията dict(), в която подаваме като аргумент последователност от двойки. Вижте примерите по-долу.

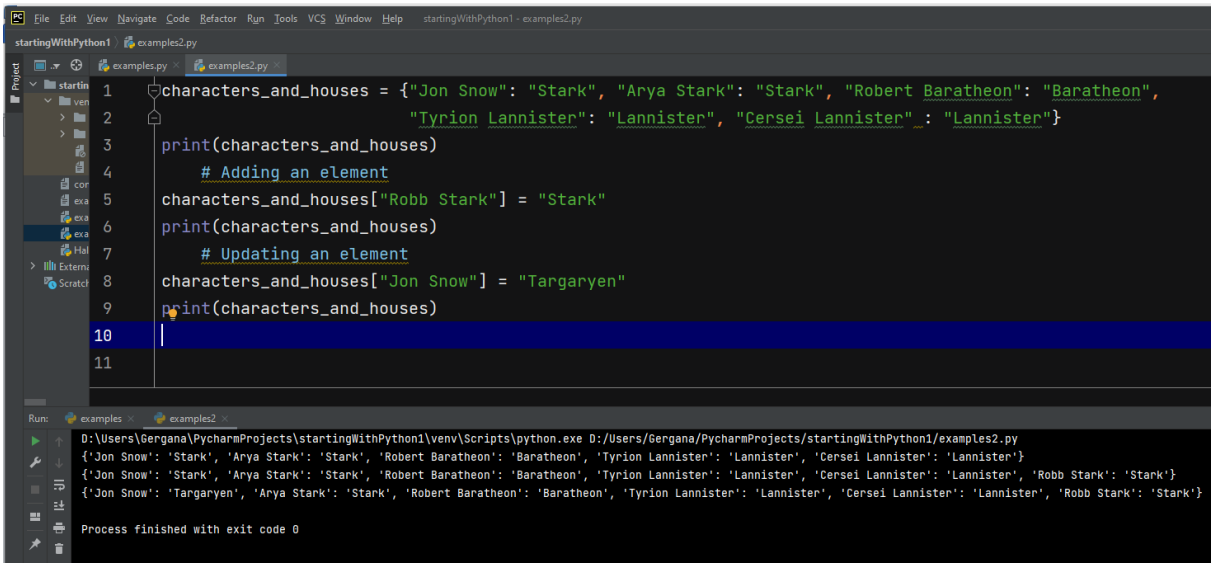


Ключът трябва да е тип данни, който е неизменен (низ, число или кортеж с неизменни елементи). Различните ключове в един и същ речник обаче могат да бъдат от различни типове данни. Те трябва да са уникални; ако добавим към речника двойка с ключ, който е идентичен с вече съществуващ ключ, то тогава вече съществуващата двойка ключ-стойност ще бъде презаписана / актуализирана (т.е. стойността, свързана с оригиналния ключ, ще бъде заменена с новопредоставената стойност за този ключ. Стойностите в двойките ключ-стойност могат да бъдат от всякакъв тип данни и могат да се дублират.

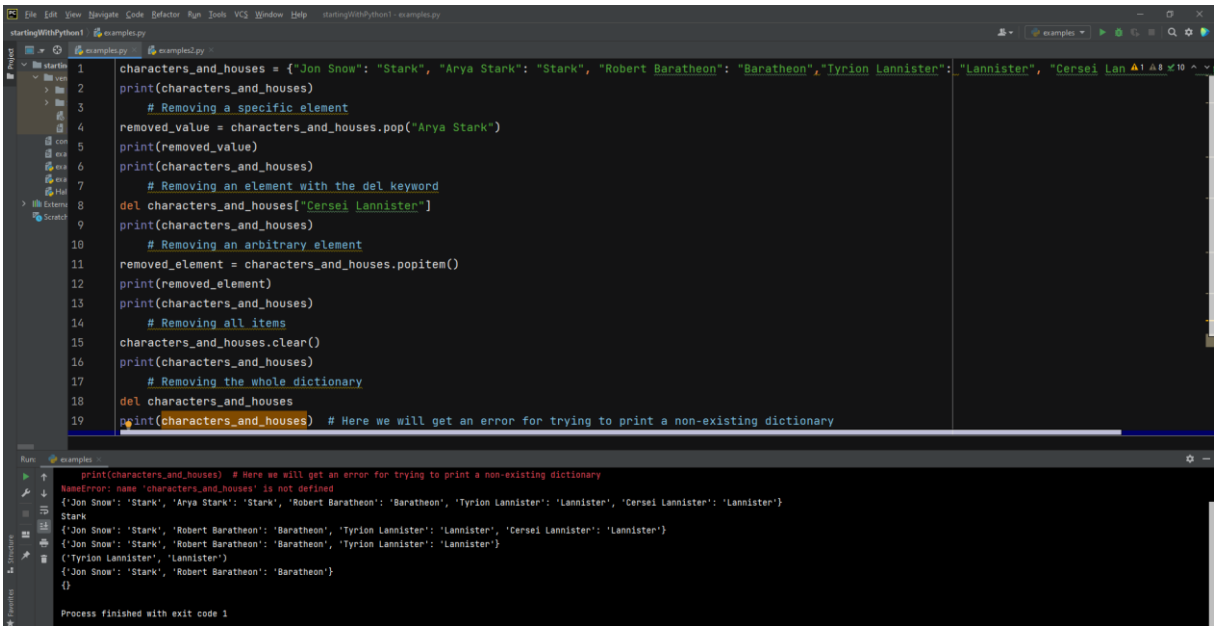
В речниците не използваме индекси за достъп до елементите, а ключове. Можем да добавяме нови елементи в речника или да променяме стойността на съществуващите елементи. Синтаксисът за добавяне и за актуализиране е един и същ – *име_речник*[*ключ*] = *стойност*. Забелязвате, че вместо да поставим индекса в квадратните скоби, ние поставяме ключа там. Вижте примерите по-долу.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



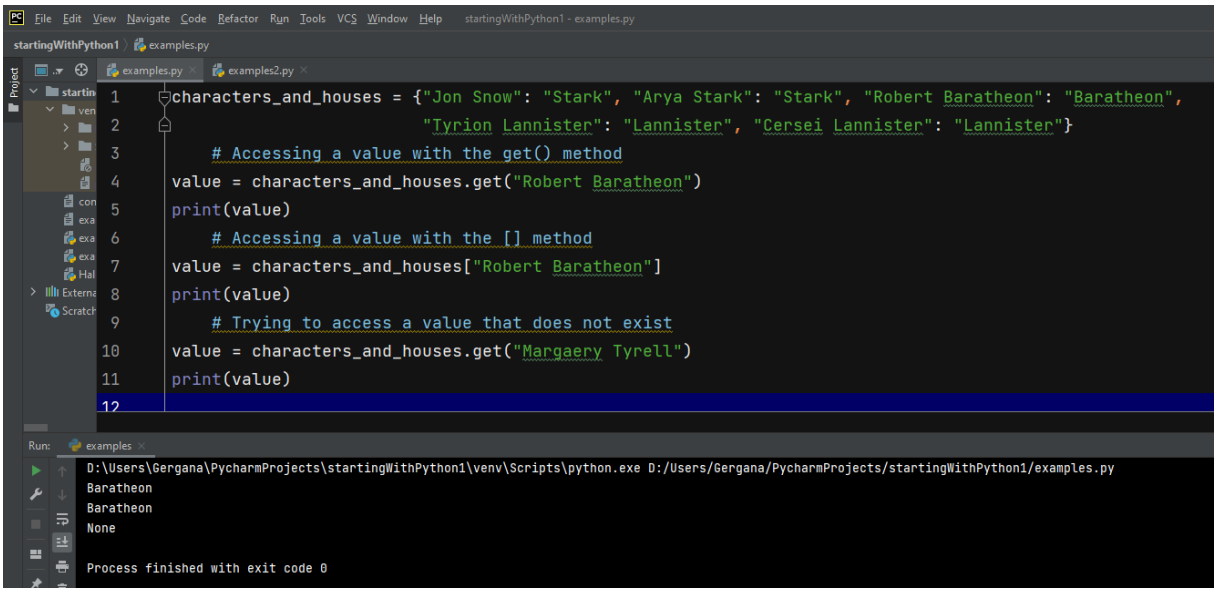
Можем да премахваме елементи от речника. Има различни методи за постигане на това - pop(), popitem() и del. Кой метод ще използваме зависи от това какво искаме да постигнем в нашата програма. Методът pop(*ключ*) премахва определен елемент (посочваме ключа, който ще бъде премахнат) и връща *стойността* му, което означава, че можем да използваме тази стойност в програмата. Методът popitem() премахва произволна двойка ключ-стойност и я връща. Ключовата дума del може да премахне отделен елемент (del *име_речник*[*ключ*]) или самия речник. Всички елементи от речника могат да бъдат премахнати наведнъж с метода clear(), като в този случай речникът ще остане празен. Вижте примерите по-долу.



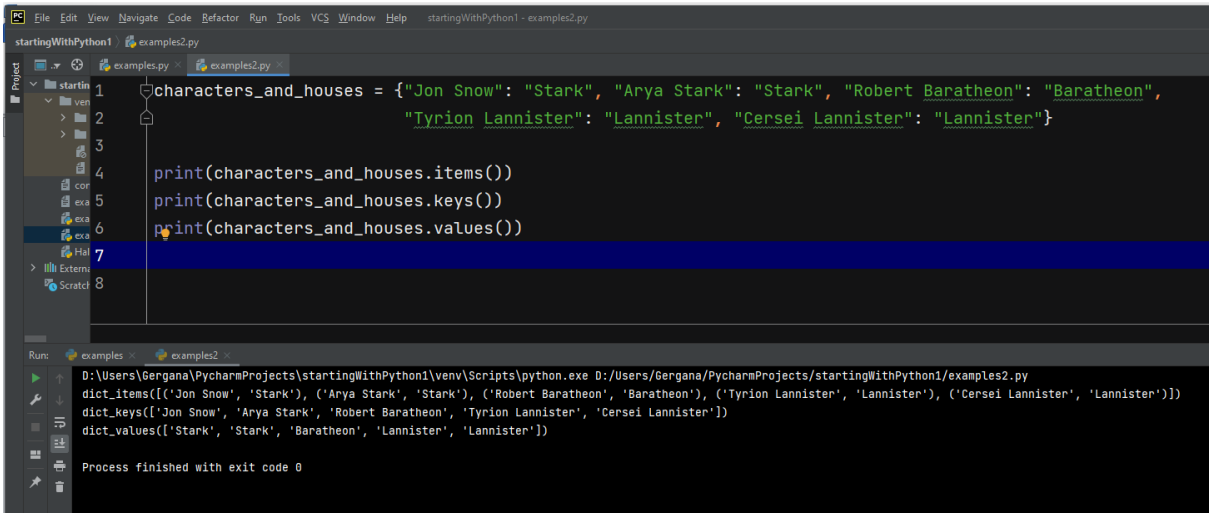
За да получите стойността на определен ключ, използвайте get(*ключ*). Ако ключът не съществува, по подразбиране се връща стойност None. Можем също така да получим достъп до стойността с метода *име_речник*[*ключ*], както вече беше обяснено по-горе. При този метод, ако ключът не е намерен, ще получим грешка и програмата ще бъде прекратена.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



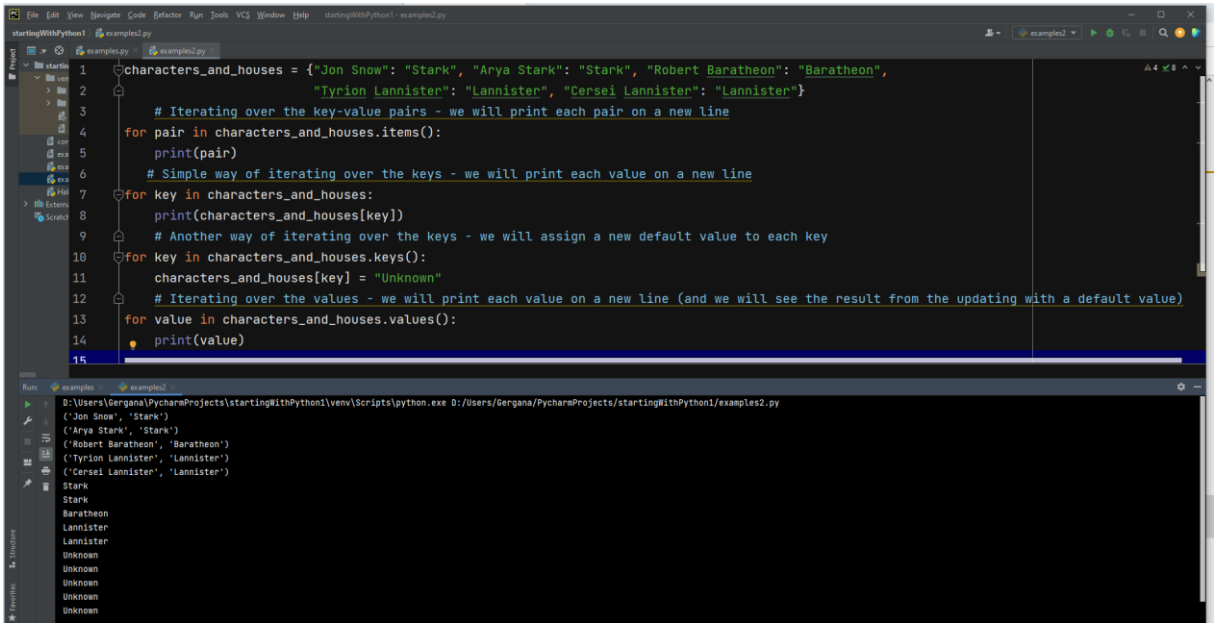
Съществуват няколко метода, които ни позволяват да получим всички двойки ключ-стойност (items()), всички стойности (values()) или всички ключове (keys()) на даден речник (например с цел да итерираме по тях). Ще забележите, че тези методи връщат списъци, съдържащи необходимата ни информация. Вижте примерите по-долу. Съществуват и други методи, които можете да научите постепенно, когато започнете да работите с речници.



Нека сега да видим как можем да итерираме по речник. Можем да итерираме по двойките ключ-стойност, по стойностите или по ключовете, като използваме резултатите от методите, показани по-горе. Съществува и опростен начин за итериране по ключовете, а именно for *key* in *име_речник*:.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

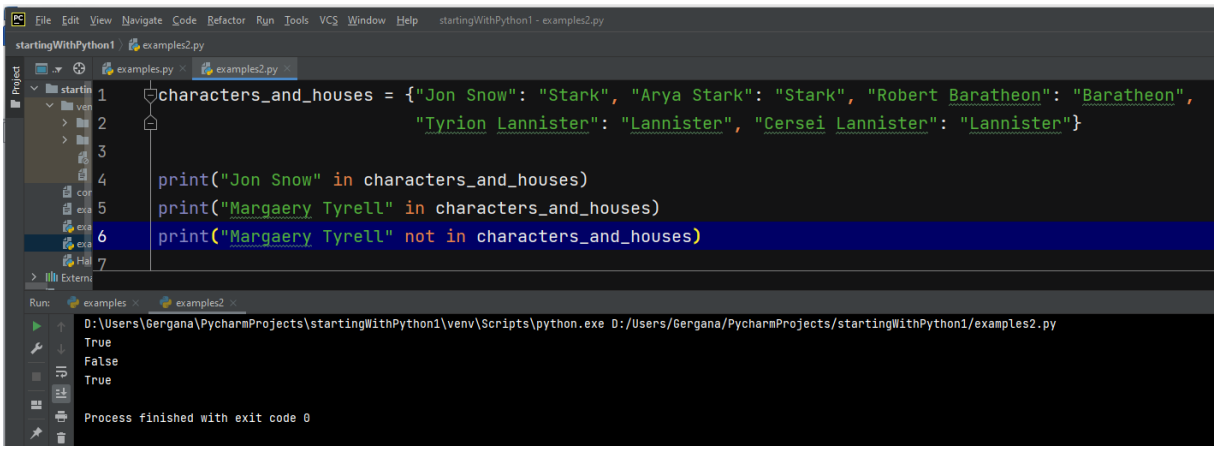
Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



```
characters_and_houses = {"Jon Snow": "Stark", "Arya Stark": "Stark", "Robert Baratheon": "Baratheon",
                         "Tyrion Lannister": "Lannister", "Cersei Lannister": "Lannister"}
    # Iterating over the key-value pairs - we will print each pair on a new line
for pair in characters_and_houses.items():
    print(pair)
  # Simple way of iterating over the keys - we will print each value on a new line
for key in characters_and_houses:
    print(characters_and_houses[key])
    # Another way of iterating over the keys - we will assign a new default value to each key
for key in characters_and_houses.keys():
    characters_and_houses[key] = "Unknown"
    # Iterating over the values - we will print each value on a new line (and we will see the result from the updating with a default value)
for value in characters_and_houses.values():
    print(value)
```

```
D:\Users\Gergana\PycharmProjects\startingWithPython1\venv\Scripts\python.exe D:/Users/Gergana/PycharmProjects/startingWithPython1/examples2.py
('Jon Snow', 'Stark')
('Arya Stark', 'Stark')
('Robert Baratheon', 'Baratheon')
('Tyrion Lannister', 'Lannister')
('Cersei Lannister', 'Lannister')
Stark
Stark
Baratheon
Lannister
Lannister
Unknown
Unknown
Unknown
Unknown
Unknown
```

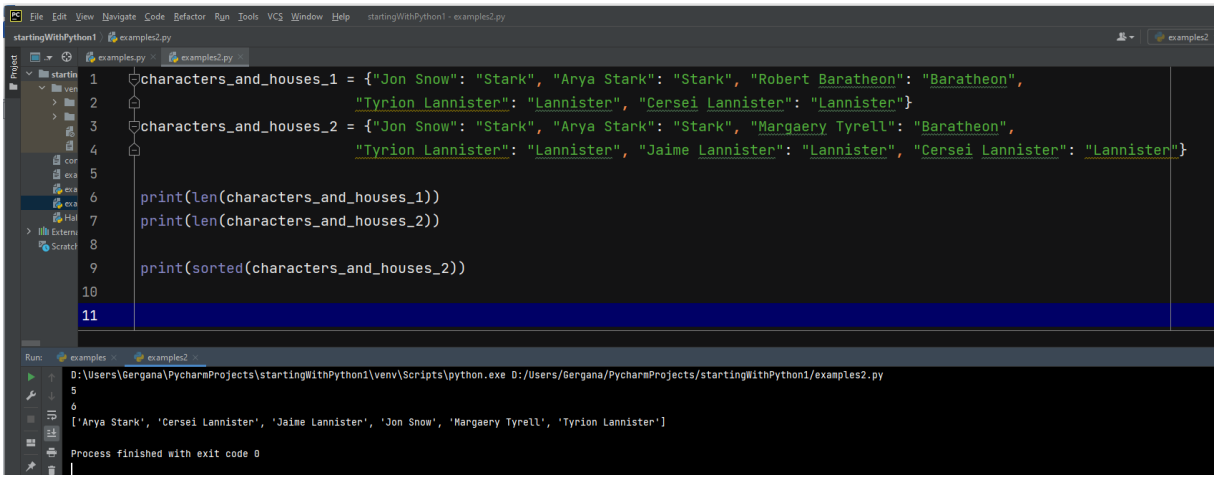
Понякога е полезно да се провери дали даден ключ присъства в речника. Това се прави с помощта на ключовата дума in или ключовите думи not in. Синтаксисът е същият като при списъците. Програмата ще върне булева стойност - True или False.



```
characters_and_houses = {"Jon Snow": "Stark", "Arya Stark": "Stark", "Robert Baratheon": "Baratheon",
                         "Tyrion Lannister": "Lannister", "Cersei Lannister": "Lannister"}

print("Jon Snow" in characters_and_houses)
print("Margaery Tyrell" in characters_and_houses)
print("Margaery Tyrell" not in characters_and_houses)
```

```
D:\Users\Gergana\PycharmProjects\startingWithPython1\venv\Scripts\python.exe D:/Users/Gergana/PycharmProjects/startingWithPython1/examples2.py
True
False
True

Process finished with exit code 0
```

Речниците в Python имат няколко полезни вградени функции, като len() (връща броя на елементите в речника) и sorted() (сортира ключовете в речника и ги връща като списък).



```
characters_and_houses_1 = {"Jon Snow": "Stark", "Arya Stark": "Stark", "Robert Baratheon": "Baratheon",
                           "Tyrion Lannister": "Lannister", "Cersei Lannister": "Lannister"}
characters_and_houses_2 = {"Jon Snow": "Stark", "Arya Stark": "Stark", "Margaery Tyrell": "Baratheon",
                           "Tyrion Lannister": "Lannister", "Jaime Lannister": "Lannister", "Cersei Lannister": "Lannister"}

print(len(characters_and_houses_1))
print(len(characters_and_houses_2))

print(sorted(characters_and_houses_2))
```

```
D:\Users\Gergana\PycharmProjects\startingWithPython1\venv\Scripts\python.exe D:/Users/Gergana/PycharmProjects/startingWithPython1/examples2.py
5
6
['Arya Stark', 'Cersei Lannister', 'Jaime Lannister', 'Jon Snow', 'Margaery Tyrell', 'Tyrion Lannister']

Process finished with exit code 0
```

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

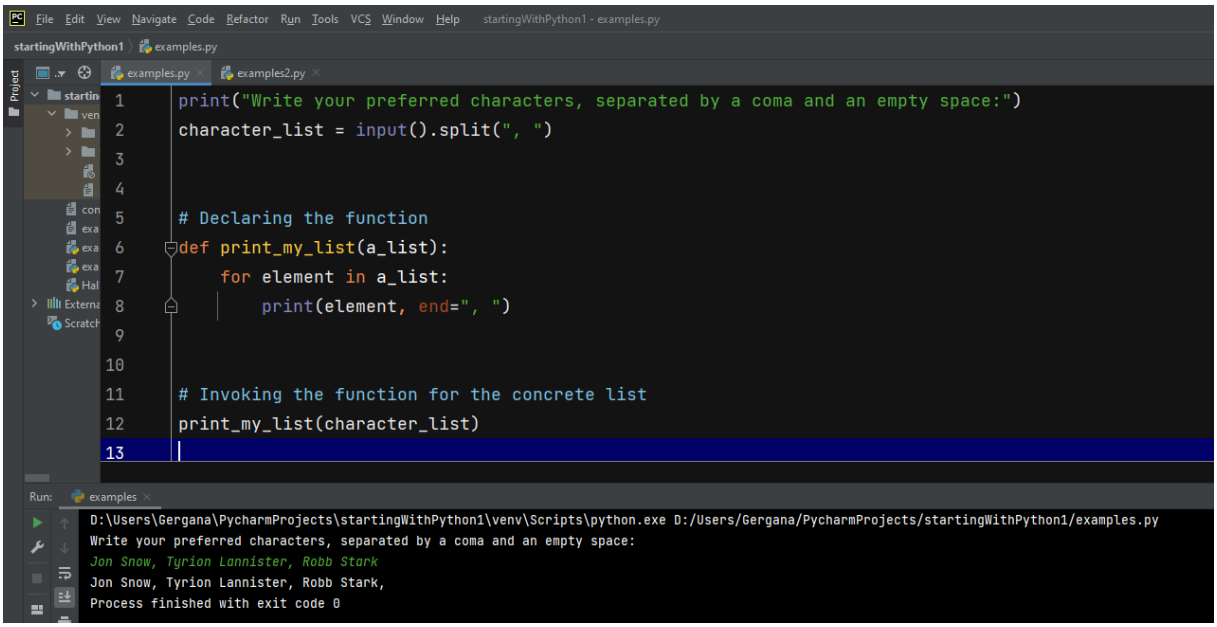
Функции

Функциите в Pyhton са блокове от код, които изпълняват определена задача. Те получават име и могат да бъдат *извиквани* чрез това име. Те са полезни, защото разбиват кода на програмата на по-малки, по-лесни за разбиране и многократно използваеми модули. Функциите могат да приемат параметри и да връщат резултат. Съществуват обаче функции, които не приемат параметри и/или не връщат резултат (ако функцията не връща резултат, то тя само изпълнява код, например отпечатва).

Функциите се декларират с ключовата дума def и тяхното име, например def *print_my_orders*:. Ако функцията ще приема *параметри,* те се добавят в скоби след името. Изпълнимият блок от код идва след : (двоеточие), на нов ред, с отстъпи. Декларираната функция е като шаблон за малка подпрограма. Тя не прави нищо, докато не бъде извикана в основната програма и не получи конкретни аргументи, ако е приложимо.

След като бъде декларирана, функцията може да бъде извикана чрез нейното име, последвано от скоби, в които се подават *аргументи* (освен ако функцията не приема параметри). Следователно *параметрите* са променливите, включени в скобите, когато декларираме функцията, а *аргументите* са конкретните стойности на тези променливи, подадени във функцията, когато тя се извиква.

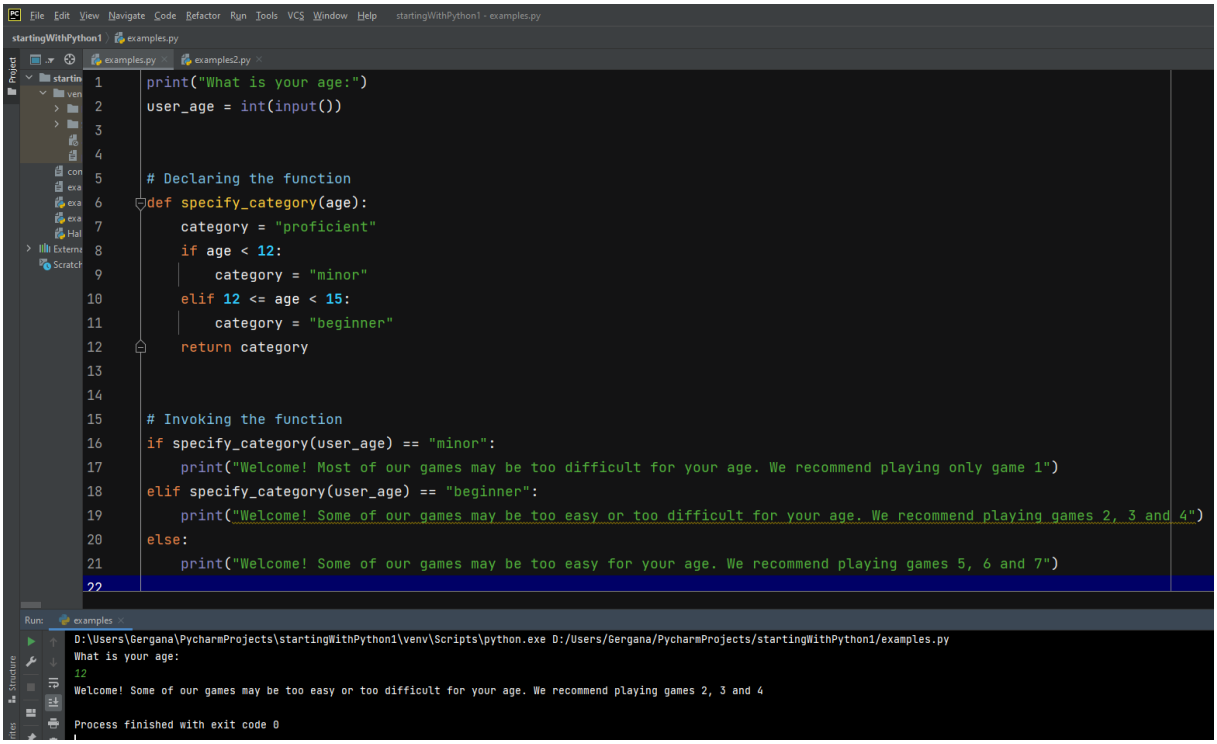
В примера по-долу функцията просто отпечатва и не връща нищо. Забележете, че параметърът a_list е променлива, на която сме дали общо име при декларирането на функцията - това не е конкретен списък, който може да използваме. Когато извикаме функцията, ще й подадем като *аргумент* конкретен списък и функцията ще изпълни с него посочените задачи. По този начин функцията може да бъде извикана многократно за различни списъци и винаги ще изпълнява същите задачи за конкретния списък, който сме подали като аргумент. Ето защо функциите правят кода многократно използваем. Не е необходимо да пишем този код отново и отново всеки път, когато ни е необходимо да изпълним тази задача, а просто извикваме функцията и й подаваме конкретния списък като аргумент. Аргументът, който подаваме във функцията, когато я извикваме, трябва да е от същия тип данни като параметъра, който сме предвидили при декларирането на функцията. Макар че Python не изисква от нас да посочваме типа данни, функцията може да не работи, ако типът данни е неправилен.



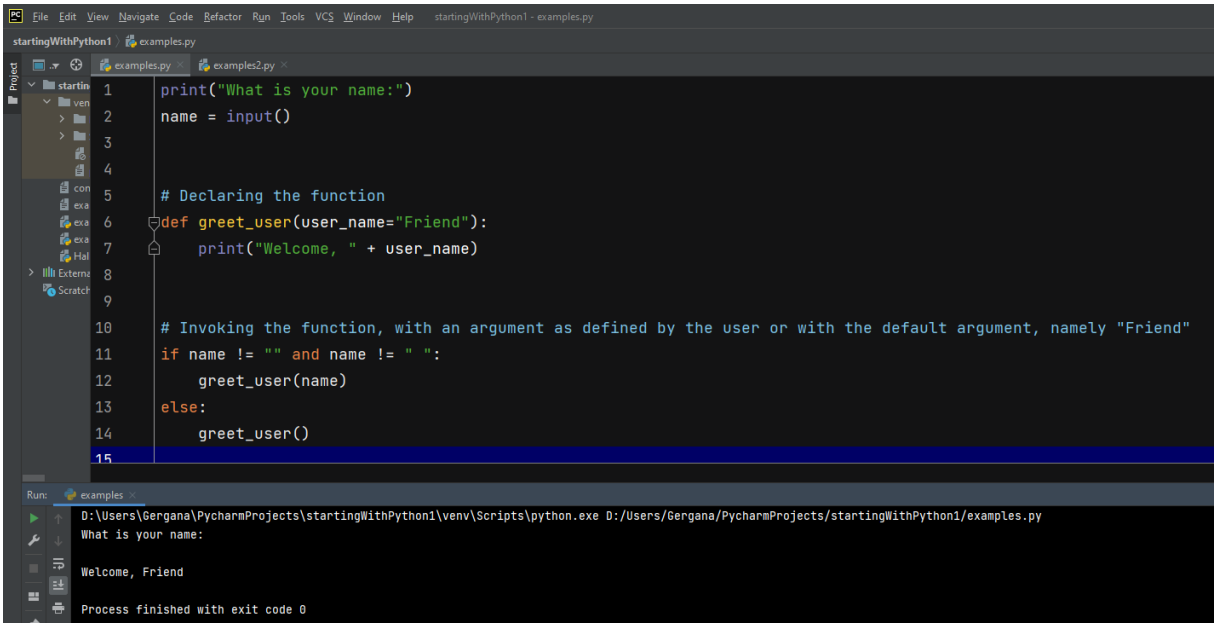
Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

Вижте също пример за проста функция, която връща резултат обратно на програмата, която след това използва този резултат според нуждите си. След това се опитайте да измислите своя собствена програма, която декларира и извиква функция, и проверете дали тя работи правилно.



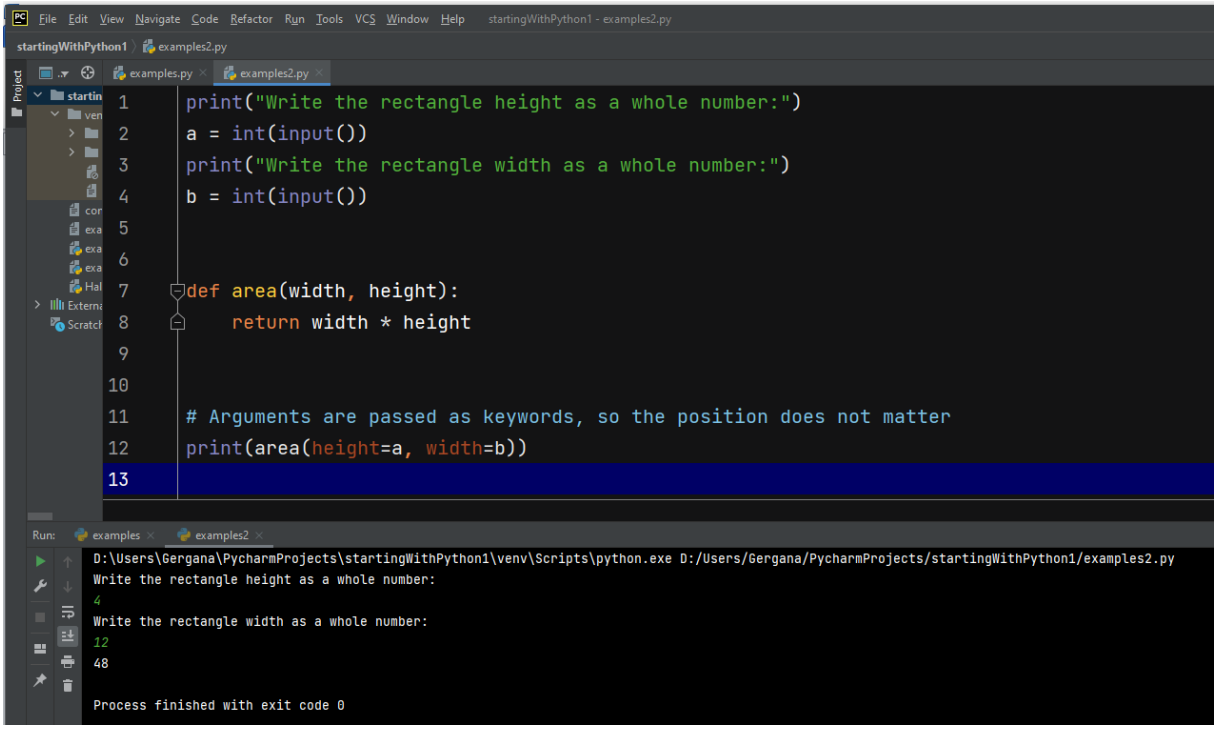
Параметърът може да има стойност по подразбиране, която ще бъде приета като аргумент, ако функцията бъде извикана без аргумент. В примера по-долу, ако потребителят не напише името си, ще го поздравим просто като “Приятелю”. Това прави програмата по-подходяща за различно поведение на потребителя, например в случаите, в които той не напише нищо на конзолата.



Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

В Python можем да използваме и функции с така наречените аргументи от ключови думи, които се именуват точно както параметрите. В този случай редът, в който подаваме аргументите, не е важен, защото те ще бъдат идентифицирани по името си.



Обекти и класове

Обектите се използват широко във всички езици за програмиране в парадигмата на програмиране, наречена обектно-ориентирано програмиране (ООП). При ООП свързани по между си свойства (атрибути) и поведения се обединяват в обекти. Програмата поддържа няколко обекта, които могат да взаимодействат помежду си. Обектите решават много проблеми в програмирането и заедно с функциите позволяват кодът да се използва многократно, вместо непрекъснато да се преписва.

За да приложим ООП в Python, трябва да познаваме няколко концепции и да научим някои синтаксиси, които всъщност са доста сходни с това, което вече знаем от други езици за програмиране.

Класове

Класът е *шаблон* за създаване на обекти. Създаването на нов клас създава нов тип обект, като позволява създаването на нови инстанции (екземпляри) от този тип. Класът предоставя описание на обекта, например какви са основните му атрибути и какво е поведението му. Всеки обект, който по-късно създаваме въз основа на този шаблон, всъщност е конкретна инстанция (екземпляр) на своя клас, с конкретни стойности за атрибутите (свойствата). Класовете съдържат атрибути на класа (общи за всички инстанции) и атрибути на инстанцията. Класовете могат да включват методи на класа за модифициране на състоянието на класа и методи на инстанцията за модифициране на инстанцията.

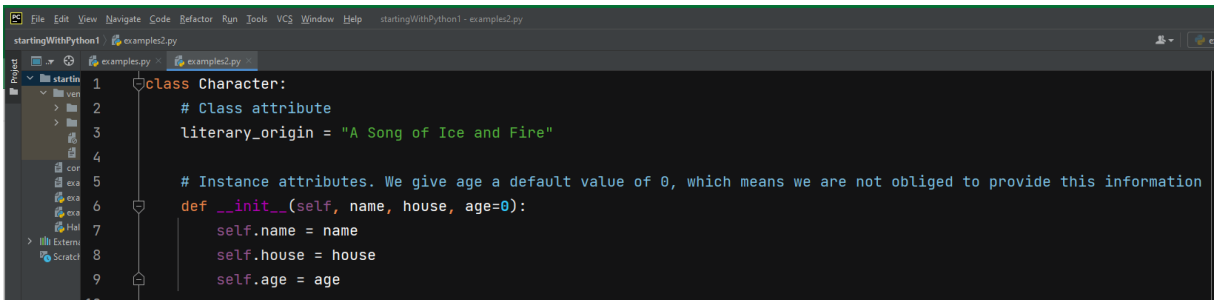
Методът __init__() инициализира началните атрибути на обекта (записва се с двойно водещо и завършващо подчертаване).

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

Параметърът self е препратка към текущата инстанция на класа. Използваме го за достъп до променливи, които принадлежат към този клас, и винаги е първият параметър, който се подава в методите на инстанция и във __init__ функцията.

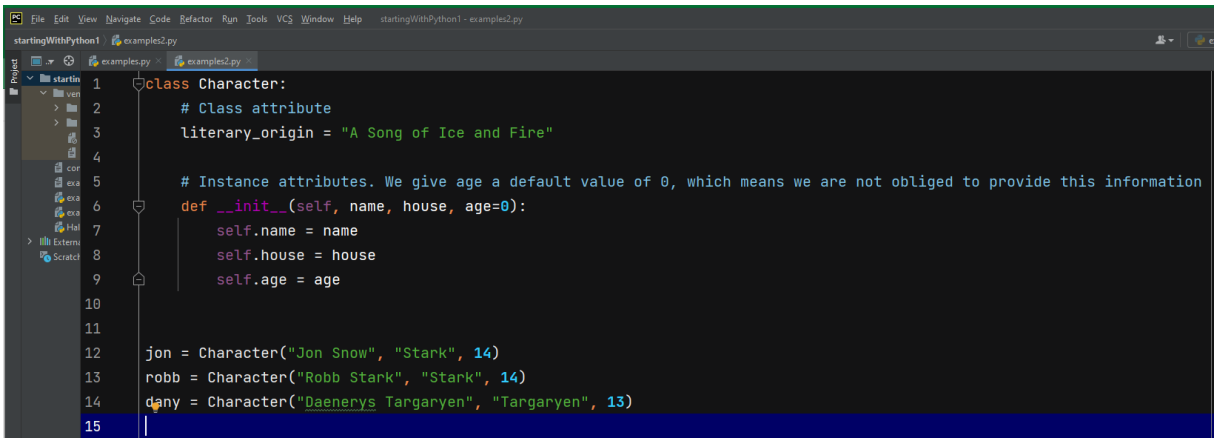
Ако всичко това ви звучи твърде абстрактно, разгледайте примера по-долу. Създаваме клас Character. Всяка инстанция (всеки конкретен екземпляр) ще има три атрибута - име, Дом и възраст. Това са атрибутите на инстанцията на класа и те също са параметри във __init__ функцията. Когато създаваме инстанция, функцията __init__ ще очаква 3 аргумента - за името, за възрастта и за Дома. Класът има атрибут literary_origin, който е "Песен за огън и лед" за всички инстанции на този клас. За атрибутите, които не са задължителни, трябва да дадем стойност по подразбиране и те трябва да бъдат посочени като последни параметри във __init__ функцията. В нашия случай даваме възраст по подразбиране 0.



```
class Character:
    # Class attribute
    literary_origin = "A Song of Ice and Fire"

    # Instance attributes. We give age a default value of 0, which means we are not obliged to provide this information
    def __init__(self, name, house, age=0):
        self.name = name
        self.house = house
        self.age = age
```

Инстанции на класове

Досега научихме, че класът е празен шаблон, а инстанцията е копие (екземпляр) на класа с действителни стойности. Сега сме готови да създадем толкова инстанции на този клас, колкото ни трябват, като му подадем конкретните имена, Домове и възраст като аргументи за всяка инстанция. Инстанцията на класа е обект. По-долу създаваме 3 конкретни персонажа въз основа на шаблона на класа. Това става чрез инициализиране на обект с *име_на_класа*(атрибути).

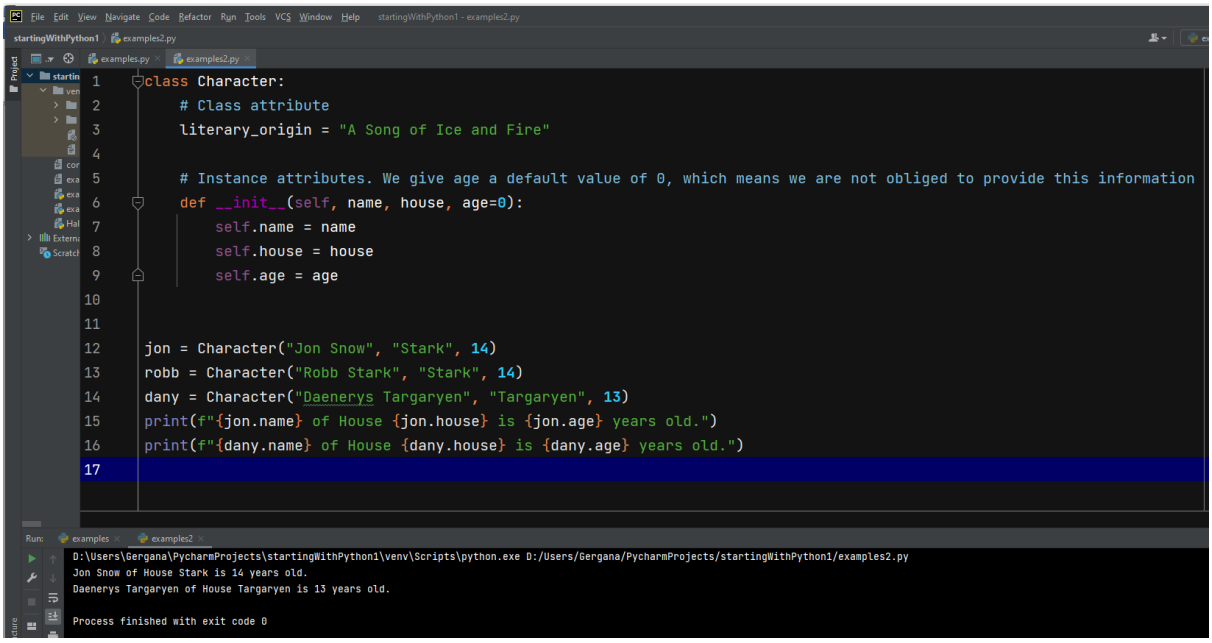


```
class Character:
    # Class attribute
    literary_origin = "A Song of Ice and Fire"

    # Instance attributes. We give age a default value of 0, which means we are not obliged to provide this information
    def __init__(self, name, house, age=0):
        self.name = name
        self.house = house
        self.age = age


jon = Character("Jon Snow", "Stark", 14)
robb = Character("Robb Stark", "Stark", 14)
dany = Character("Daenerys Targaryen", "Targaryen", 13)
```

За да получим достъп до атрибутите на конкретен обект, изписваме името на обекта, последвано от точка и името на атрибута. Например, можем да получим достъп до възрастта на Роб чрез robb.age. Вижте примера по-долу и забележете резултата. Програмата извлича атрибутите на нашите персонажи.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

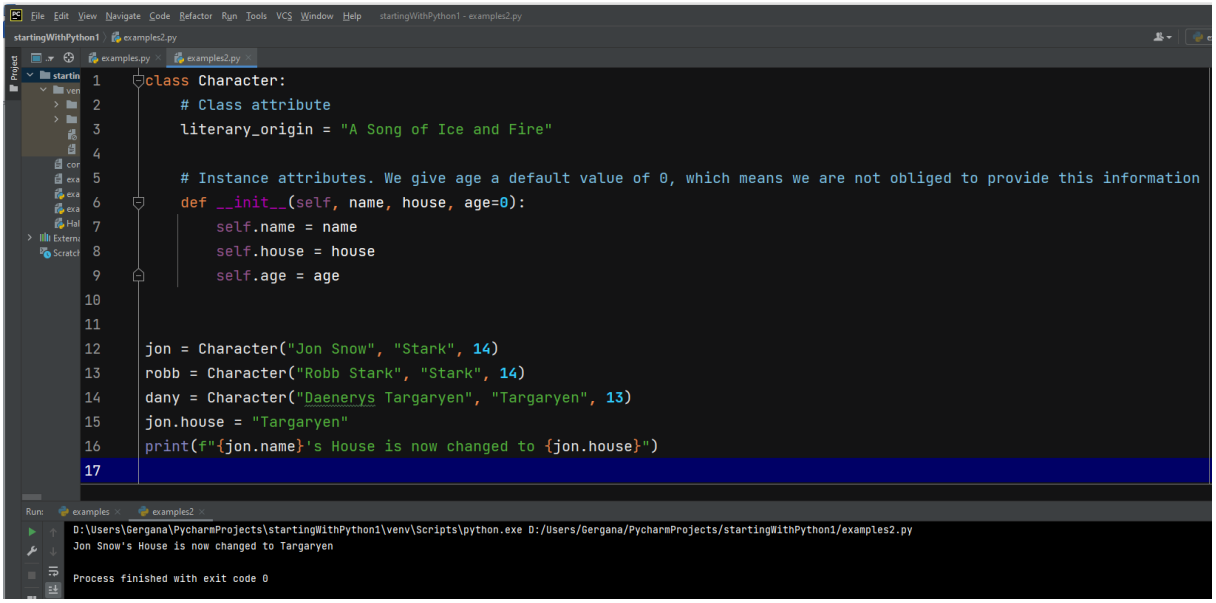
Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



```
class Character:
    # Class attribute
    literary_origin = "A Song of Ice and Fire"

    # Instance attributes. We give age a default value of 0, which means we are not obliged to provide this information
    def __init__(self, name, house, age=0):
        self.name = name
        self.house = house
        self.age = age


jon = Character("Jon Snow", "Stark", 14)
robb = Character("Robb Stark", "Stark", 14)
dany = Character("Daenerys Targaryen", "Targaryen", 13)
print(f"{jon.name} of House {jon.house} is {jon.age} years old.")
print(f"{dany.name} of House {dany.house} is {dany.age} years old.")
```

```
D:\Users\Gergana\PycharmProjects\startingWithPython1\venv\Scripts\python.exe D:/Users/Gergana/PycharmProjects/startingWithPython1/examples2.py
Jon Snow of House Stark is 14 years old.
Daenerys Targaryen of House Targaryen is 13 years old.

Process finished with exit code 0
```

След като получим достъп до атрибут на обекта, можем да го модифицираме, като му присвоим нова стойност. Вижте примера.



```
class Character:
    # Class attribute
    literary_origin = "A Song of Ice and Fire"

    # Instance attributes. We give age a default value of 0, which means we are not obliged to provide this information
    def __init__(self, name, house, age=0):
        self.name = name
        self.house = house
        self.age = age


jon = Character("Jon Snow", "Stark", 14)
robb = Character("Robb Stark", "Stark", 14)
dany = Character("Daenerys Targaryen", "Targaryen", 13)
jon.house = "Targaryen"
print(f"{jon.name}'s House is now changed to {jon.house}")
```

```
D:\Users\Gergana\PycharmProjects\startingWithPython1\venv\Scripts\python.exe D:/Users/Gergana/PycharmProjects/startingWithPython1/examples2.py
Jon Snow's House is now changed to Targaryen

Process finished with exit code 0
```

Можем да получим достъп до атрибутите на класа по подобен начин. Вижте по-долу.

Съвет за форматиране на низове:

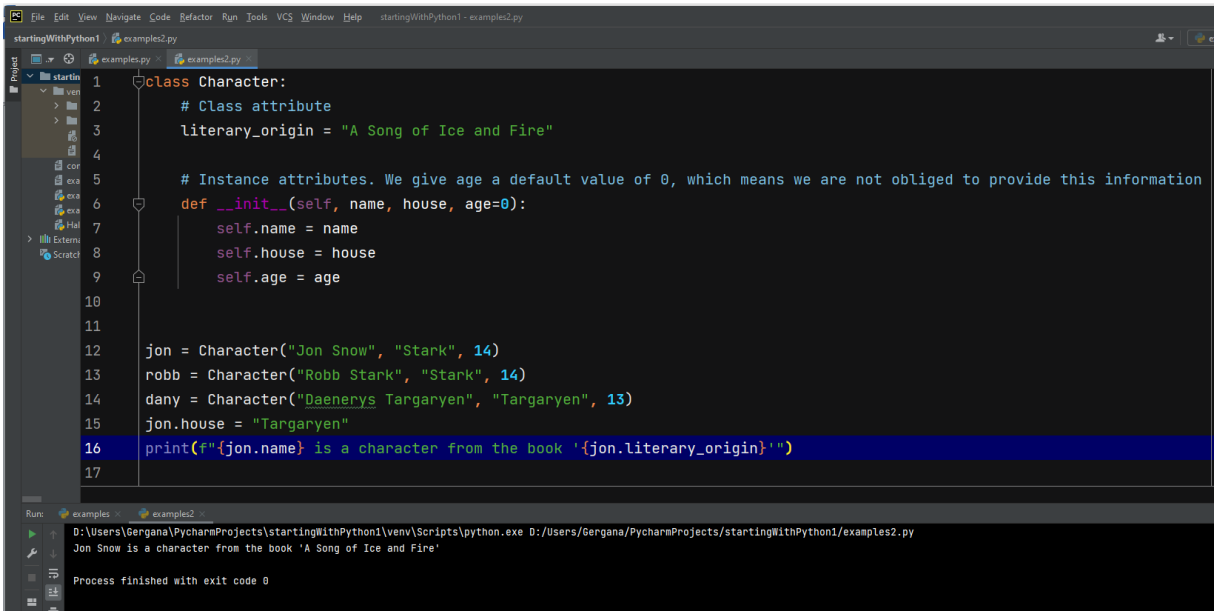
Ако искате да напишете кавички във форматиран низ, имате две възможности:

Едната от тях е да използвате формат на кавичките, който е различен от формата, който обикновено използвате за низове. По-долу създаваме низове с двойни кавички, затова използваме единични кавички вътре в текста.

Друга възможност е да "избегнем" (escape) символа, като напишем \ преди него. По този начин Python ще разбере, че той не е част от кода, а трябва да бъде отпечатан като специален символ. Можете да "избегнете" всеки специален символ по този начин. Опитайте сами.

Съфинансиран от програма „Еразъм+“ на Европейския съюз

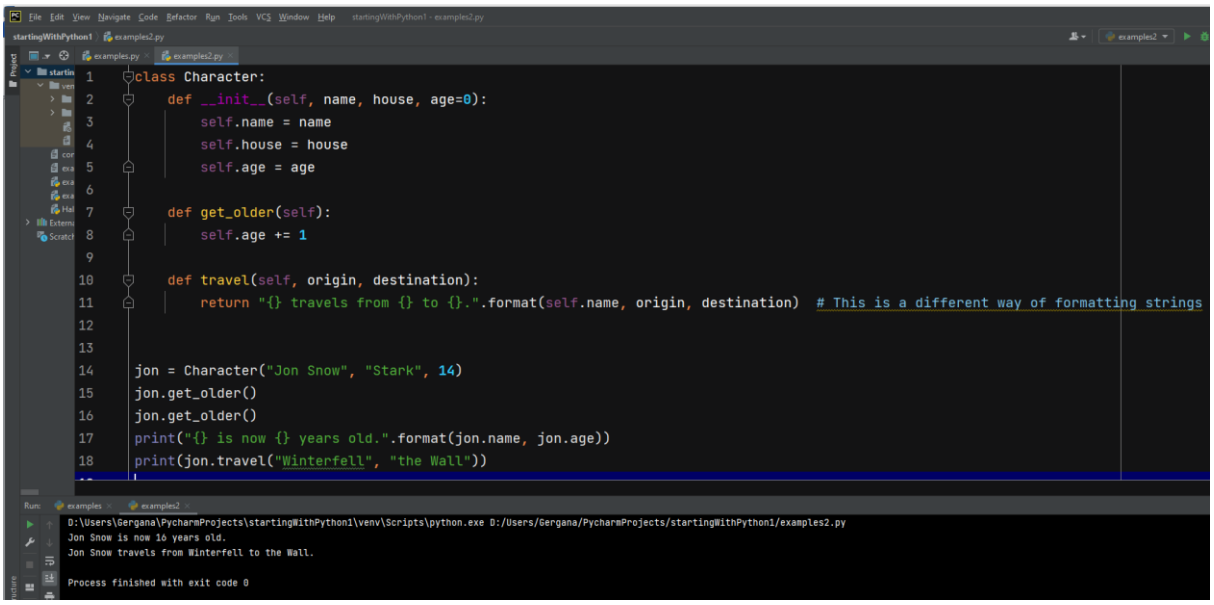
Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.



```
class Character:
    # Class attribute
    literary_origin = "A Song of Ice and Fire"

    # Instance attributes. We give age a default value of 0, which means we are not obliged to provide this information
    def __init__(self, name, house, age=0):
        self.name = name
        self.house = house
        self.age = age


jon = Character("Jon Snow", "Stark", 14)
robb = Character("Robb Stark", "Stark", 14)
dany = Character("Daenerys Targaryen", "Targaryen", 13)
jon.house = "Targaryen"
print(f"{jon.name} is a character from the book '{jon.literary_origin}'")
```

```
D:\Users\Gergana\PycharmProjects\startingWithPython1\venv\Scripts\python.exe D:/Users/Gergana/PycharmProjects/startingWithPython1/examples2.py
Jon Snow is a character from the book 'A Song of Ice and Fire'

Process finished with exit code 0
```

Нека сега разгледаме методите на инстанциите. Те се дефинират вътре в класа, но се използват за получаване на съдържанието на инстанцията или за извършване на действия с атрибутите на инстанцията. Обикновено те определят поведение. Първият им параметър винаги трябва да бъде self, както във функцията __init__. В примера по-долу създаваме два метода за нашите персонажи - get_older() и travel(). Методът get_older() състарява персонажа с 1 година, а методът travel() връща текст, определящ откъде пътува и закъде пътува. Когато извикваме тези методи в програмата, ги извикваме по същия начин, по който получаваме достъп до атрибути - *име на обекта,* последвано от *.име_на_метода* - например jon.travel(). Също така трябва да подадем всички аргументи, които методът очаква. Методът get_older() не изисква аргумент, тъй като винаги състарява персонажа с 1 година. Методът travel() изисква като аргументи началната и крайната точка на пътуването. Можете да модифицирате метода get_older(), така че да увеличава възрастта на персонажа със зададен брой години. В този случай трябва да включите това число като параметър при дефинирането на метода и след това да го подадете като аргумент при извикването на метода. Опитайте това сами.



```
class Character:
    def __init__(self, name, house, age=0):
        self.name = name
        self.house = house
        self.age = age

    def get_older(self):
        self.age += 1

    def travel(self, origin, destination):
        return "{} travels from {} to {}.".format(self.name, origin, destination)  # This is a different way of formatting strings


jon = Character("Jon Snow", "Stark", 14)
jon.get_older()
jon.get_older()
print("{} is now {} years old.".format(jon.name, jon.age))
print(jon.travel("Winterfell", "the Wall"))
```

```
D:\Users\Gergana\PycharmProjects\startingWithPython1\venv\Scripts\python.exe D:/Users/Gergana/PycharmProjects/startingWithPython1/examples2.py
Jon Snow is now 16 years old.
Jon Snow travels from Winterfell to the Wall.

Process finished with exit code 0
```

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.

Заключителни бележки

Поздравления, усвоихте основите на Python! Ако сте отделили време да пресъздадете кода или да програмирате свои собствени примери, вие сте готови да започнете да работите върху по-сложни алгоритми и да решавате задачи. Влезте в HackerRank и започнете подготовката си, като преминавате постепенно от лесен към средно труден и труден режим.

Разбира се, има още много неща, които да научите в Python. Щом се сблъскате със задача, която не можете да решите, потърсете решение в интернет. Има много информация за Python. Сайтове като https://stackoverflow.com могат да ви помогнат бързо да намерите решения дори на привидно трудни проблеми. Друг полезен източник е официалната документация на Python на адрес https://docs.python.org. И накрая, в самия PyCharm, като задържите CTRL и кликнете върху определена функция, ще получите цялата документация за тази функция, което спестява много време.

Приятна работа!

Съфинансиран от програма „Еразъм+“ на Европейския съюз

Подкрепата на Европейската комисия за изготвянето на настоящата публикация не представлява одобрение на съдържанието, което отразява гледните точки само на авторите и не може да се търси отговорност от Комисията за всяка употреба, която може да бъде използвана за информацията, съдържаща се в нея.