Представяне на езика Python

д-р Филип Андонов

6 юни 2022 г.

Съдържание

Python

Python е:

- Програмен език с общо предназначение
- От високо ниво
- Основан на философията за четимост на кода. Синтаксисът на Python позволява реализацията на алгоритми с по-малко код, отколкото на C например
- Обектно-ориентиран

Python

Python е:

- Императивен
- С динамични типове
- С автоматично управление на паметта
- С голяма стандратна библиотека

Философия на Python

Основата на философията на Python е идеята за четимост и елегантност на кода. Синтаксисът на Python позволява реализацията на алгоритми с по-малко код, отколкото например на C. Други елементи на философията на Python са, че играчките трябва да идват с батерии (за което ще стане въпрос след малко) и че живота не трябва да се взима много на сериозно (името на езика идва от Monthy Python все пак).

Философия на Python

В самият интерпретатор е вградено „дзен“ послание, което може да се прочете като се напише import this. То гласи:

- Красивото е по-добро от грозното.
- Явното е по-добро от неявното.
- Простото е по-добро от сложното.
- Сложното е по-добро от усложненото.
- Плоското е по-добро от вложеното.
- Разпръснатото е по-добро от наблъсканото.
- Четимостта е важна.
- Особените случаи не са толкова особени, че да нарушават правилата.
- Въпреки че практичността бие пуризма.
- Грешките никога не трябва да отминават тихо.

Философия на Python

- Освен ако не са нарочно заглушени.
- В случай на двусмислие, не се поддавай на изкушението да налучкваш.
- Трябва да има един – и най-добре точно един – начин да се направи нещо.
- Въпреки че това няма да е очевидно на пръв поглед, ако не си холандец.
- Сега е по-добре от никога.
- Въпреки че никога, е по-добре от веднага.
- Ако реализацията е трудна за обяснение, то идеята е лоша.
- Ако реализацията е лесна за обяснение, то може би идеята е добра.
- Пространствата от имена са страхотна идея – нека имаме повече от тях.

Употреба на Python

Големи организации, ползващи Python включват:

- Google
- CERN
- NASA
- Industrial Light and Magic

Употреба на Python

Adam Geitgey, Director of Software Engineering at Groupon

"Python is the most popular programming language today for machine learning"

Употреба на Python

Според Bjarne Stroustrup (създателят на C++) всеки програмист трябва да знае поне 5 езика от които първите три са:

- C++
- Java
- Python

Употреба на Python

Създадени са множество библиотеки, позволяващи използването на Python в научни изчисления. Някои от тях са:

- NumPy
- SciPy
- Matplotlib

Употреба на Python

Едноплатковият компютър Raspberry Pi използва Python като основен програмен език.

Инсталиране на Python

В много ОС Python е стандартен компонент и може да бъде използван от командния ред, като например повечето Linux/Unix дистрибуции, както и FreeBSD, NetBSD, OpenBSD, MacOS.

Hello world

Първата програма на всеки език от високо ниво е извеждането на текста Hello world на екрана (при вградените системи това е накрая на учебника, защото трябва сами да си напишем драйвера за екрана). Можем да използваме обема на кода, необходим за да изпълним тази простичка задача като своеобразна метрика колко стегнат в изказа си е езикът (т. нар. fluff code). Python се приближава до теоретичния минимум в този случай. Просто пишем командата и ѝ казваме какво да изведе на екрана.

```
1 print("Hello world")
```

Команден интерпретатор

Кодът, написан на Python може да бъде изпълнен по два начина. Първият е да стартираме командния интерпретатор и да пишем команди ред по ред. Това е удобно, когато искаме да пробваме някоя функция, която току-що сме научили, но не и за писането на по-дълъг код. Другият вариант, разбира се, е да използваме текстов редактор или среда за програмиране и да стартираме кода след написването му.

Команден интерпретатор

```
1  Python 3.6.9 (default, Nov  7 2019, 10:44:02)
2  [GCC 8.3.0] on linux
3  Type "help", "copyright", "credits" or "license"
       for more information.
4  >>>
5  >>> 2+2
6  4
7  >>> 34287+2348972
8  2383259
9  >>> 1/2
10 0
11 >>> 1.0/2.0
12 0.5
```

Присвояване

Присвояването в езика Python е от цитиращ (а не копиращ, както в някои други езици) тип: фактически се копира или присвоява препратка към съответната стойност, а не самата тя. Това има значение най-вече за променливите от изменим тип, за които ще стане дума по-късно. Достатъчно е да запомним, че присвояването никога не прави копие на стойност, а само на референция.

Типове данни

Променливите в Python имат тип. За всеки тип има и функция със същото име, която преобразува получения аргумент в стойност от този тип.

None

Това всъщност е обект, който се връща от функции, които нямат явна клауза return. Той не поддържа операции. Съществува само един такъв обект, който именно се казва None. Използва се, за да укаже „нищо”.

bool

В Python има отделен тип за булеви стойности, наречен bool. Той е подтип да целочисления тип. Обектите от този тип могат да заемат само една от две стойности - True и False. За разлика от езиците, които са взаимствали синтаксиса на логическите операции от C, в Python те са and, or и not съответно за логическо И, ИЛИ и НЕ.

Обикновените целочислени променливи не могат да приемат стойности, по-големи от 2147483647 (или по-малки от –2147483648).

long

Ако искаме по-големи целочислени стойности, използваме типа LONG. Неговите литерали се пишат също като нормалните целочислени стойности, но с L накрая. (Можем да използваме и долен регистър, но тогова символът прилича на този за единица и затова е по-добре да се избягва). Те са ограничени единствено от размера на свободната памет. Шестнайсетичните числа се пишат, както е показано на ред 3, а осмичните, както на ред 5.

Listing 1: Longs, осмични, шестнайсетични

```
1 >>> 10000000000000000L /2
2 5e+17
3 >>> 0xAF
4 175
5 >>> 010
6 8
```

Аритметически операции

Основните аритметически операции и знаците, с които се извършват, са дадени в таблицата по-долу.

Таблица: Символи за аритметически операции

Оператор	Операция	Пример	Резултат
+	събиране	3+2	5
-	изваждане	3-2	1
*	умножение	3*2	6
/	деление	10/2	5
//	целочислено деление	10//2	5
**	степен	2**3	8
%	остатък от целочислено деление	4%3	1

float

Числата с плаваща запетая в Python са от тип float. Когато пишем стойност от този тип ние слагаме десетична точка или можем да използваме символа E за да използваме научната нотация. При нея след символа E се поставя число което указва 10 на коя степен трябва да се повдигне, преди да се умножи по стойността. В повечето платформи Python държи числата с плаваща запетая като 64-битови стойности с „двойна точност“ според стандарта IEEE 754. В този случай максималната стойност на този тип е приблизително $1.8 * 10^{308}$.

Listing 2: числа от тип float

```
1  >>> 3.14
2  3.14
3  >>> type(3.14)
4  <class 'float'>

6  >>> .1e7
7  1000000.0
8  >>> type(.1e7)
9  <class 'float'>
10 >>> 1.e-4
11 0.0001
```

Трябва да имаме предвид, че вътрешното представяне на числата с плаваща запетая е резултат от деление на две числа (рационални числа). Така че от много малка част от реалните числа могат да бъдат представени точно с този тип. Останалите са стойности, близки до истинската. В повечето случаи това не е проблем, но все пак не бива да го забравяме.

Символни низове

В Python можем да използваме както апострофи, така и кавички, за да оградим символен низ. Ако символите, с които работим, са литерали, то единствената причина да сменяме символа за ограждане е, ако той се съдържа в низа. Ако искаме да имаме апостроф, то е по-добре да оградим низа с кавички. Многоредовите символни низове се ограждат с три апострофа или кавички.

Listing 3: Литерали от символен тип

```
>>> "Let's go!"
"Let's go!"
>>> '"Hello, world!" she said'
'"Hello, world!" she said'
>>> 'Let's go!'
SyntaxError: invalid syntax
>>> 'Let\'s go!'
"Let's go!"
>>> "\"Hello, world!\" she said"
'"Hello, world!" she said'
```

Символни низове

За да укажем, че искаме даден символ да се интерпретира просто като символ, а не в специален смисъл, то можем да поставим наклонена черта \ за да го направим. Това се нарича escape character. Ако искаме да напишем просто наклонена черта, тогава трябва да „escape“-нем и нея. Дотук видяхме символните низове в интерпретатора. Ако искаме да ги печатаме (например в записана програма на Python), ще използваме разгледаната вече команда print.

Символни низове

```
>>> "Hello, world!"
'Hello, world!'
>>> 10000L
10000L
>>> print("Hello, world!")
Hello, world!
>>> print(10000L)
10000
>>> temp = 42
>>> print("The temperature is " + temp)
Traceback (most recent call last):
File "<pyshell#61>", line 1, in ?
print("The temperature is " + temp)
TypeError: cannot add type "int" to string
>>> print("The temperature is " + repr(temp))
The temperature is 42
```

Вход-изход

В Python извеждането на екрана става с функцията print. Във версия 2 print беше команда и не изискваше скоби.
За четене от клавиатурата се използва функцията input. Тя приема като параметър низ, който се извежда като подканящо съобщение на потребителя. Python отново пести усилия на разработчиците. В другите езици това трябва да стане с две команди: изведи подканящо съобщение и прочети въведеното от потребителя.

Вход-изход

```
>>> meaning = input("The meaning of life: ")
The meaning of life: 42
42
>>> print("But what is the question then?")
But what is the question then?
>>>
```

Извикване на функции

Функция е подпрограма, която можем да извикаме, като ѝ подадем аргументи на входа и очакваме нещо на изхода. Често използвани групи от оператори обикновено се изнасят във функции. В Python, както в повечето езици, функцията се извиква като се изпише нейното име и в кръгли скоби се подадат аргументите. Отново те могат да бъдат както литерали, така и променливи, така и изрази, включително други функции.

Извикване на функции

```
1 >>> pow(2,3)
2 8
3 >>> abs(-10)
4 10
5 >>> 10+pow(abs(-2),3*5)/3.0
6 10932.666666666666
```

Модули

Модулите са разширения на езика, които могат да бъдат внесени в Python, за да разширят възможностите му. Внасянето на модули се извършва с import. Има няколко начина да внесем функционалност от външен модул в зависимост от това какво точно искаме да направим.

Модули

import име_на_модул – внася всичко от модула. За да можем да използваме в нашия код нещо от модула, винаги трябва да пишем името на модула, точка и след това името на декларираното в модула.

```
>>> meaning = input("The meaning of life: ")
The meaning of life: 42
42
>>> print("But what is the question then?")
But what is the question then?
>>>
```

Модули

`from module import функция_1, функция_2` – внасяме само функции 1 и 2 от модула. Обръщаме се към тях сякаш са дефинирани в нашия файл.

```
1 from math import floor, ceil
2 print floor(math.pi), ceil(math.pi)
```

Модули

`from module import *` – подобно на горното, но внася всички функции от модула. Като цяло се счита за лоша идея, защото може да има (и има) модули с еднакви имена на функции в тях.

```
1 from math import *
2 print floor(pi)
```

from module import функция_1 as псевдоним – когато искаме да внесем функция, която е с прекалено дълго име, можем да я прекръстим.

```
1 from math import factorial as fact
2 fact(5)
```

Модули

Следва непълен списък от някои стандартни модули и тяхната функционалност.

sys– предоставя функции и променливи, които могат да се използват за настройка на средата на изпълнение на Python. Например, четенето на параметри от командния ред се осъществява с функции, включени в този модул.

os– предоставя унифициран интерфейс към функции на операционната система. Повечето функции в този модул са платформено-зависими, но модулът избира да зареди коректната имплементация при внасяне.

Webbrowser– предоставя функции за показване на уеб-страници.

fileinput – имплементира спомагателен клас и функции за итериране върху множество от файлове, подадени на стандартния вход или като списък от файлове.
time– предоставя функции за работа с дати и времена. Базиран е на C библиотека. Дадена дата и час може да бъде представена като число с плаваща запетая (изразяващо броя на секундите, изминали от някакво начало, например 1 януари 1970 година) или като кортеж.

Модули

random– имплементира генератори за псевдо-случайни числа.
math– съдържа математически оператори и константи.
cmath– представлява същото като math, но и за комплексни числа.

Внасяне на модули

Нека илюстрираме с пример.

```
1  >>> import math
2  >>> math.floor(32.9)
3  32.0
4  >>> from math import sqrt
5  >>> sqrt(9)
6  3.0

8  >>> import cmath
9  >>> cmath.sqrt(-1)
10 1j
```

Python програми

Както стана ясно програмите на Python често се интерпретират, така че когато искаме да ги разпространяваме ние обикновено просто пращаме самия изходен файл. Възможно е обаче да се разпространява и компилирания файл с разширение pyc.

Python програми

Структурата на една Python програма е сравнително проста. Тя започва с внасянето на модули, ако това е необходимо. Всичко останало няма задължително място, но разбира се има ограничения. Както и в повечето други езици глобалните променливи, функциите и класовете трябва да се намират преди обръщенията към тях. В Python няма основна функция или метод, която да бъде входна точка. Първият израз, който напишем в кода ще се изпълни при стартиране.

Коментари

В Python едноредовите коментари се създават със символа # в началото на реда.
Често за коментари се използва и т. нар. docstring, което просто означава символен литерал поставен в кода и не присвоен на променлива.

```
1 #This is a comment
2 '''
3 This is a multi-line
4 comment
5 '''
```

Малки удобства

Когато функция се използва често по специфичен начин, в Python тя е създадена така, че да улеснява именно тази честа употреба. Например, когато искаме да изведем нещо на екрана, то обикновено е комбинация от литерали на символни низове и стойности на променливи. В повечето езици това се постига като се конвертират всички стойности до тип символен низ и се извърши конкатенация или се използват форматиращи символи. В Python това е решено, като print може да приема множество аргументи, които трябва да бъдат разделени със запетая.

Listing 4: Множество аргументи на print

```
1 >>> answer = 42
2 >>> question = "about the life, universe and
    everything"
3 >>> print("the answer of the question ",question,
    " is ", answer)
```

Друго удобство е синтаксисът, който позволява множествено присвояване. Отляво на символа за равенство е списък с променливи, разделени със запетая, а отдясно – съответстващите им стойности, отново разделени със запетая. Това е демонстрирано на ред 1 от кода по-долу. На редове 2, 3 и 4 е реализирана размяна на стойностите на две променливи чрез помощна трета, както се извършва в стандартните езици (например C++). На ред 5 е показано същото нещо, направено чрез множествено присвояване.

Listing 5: Множествено присвояване

```
1 a, b, c = 1, 2, 3
2 b = a
3 a = c
4 c = b
5 a, c = c, a
```

Блокове

Блок са оператори, групирани така, че да се приемат за един. Това са, например, групата оператори, които се изпълняват, ако дадено условие е вярно или се повтарят в цикъл и т.н. Ако сме писали на друг език за програмиране, очакваме блоковете да се заграждат със специални символи, например скоби {}, или ключови думи, например begin и end. В Python обаче блоковете се създават с отместване на кода навътре, поставяйки интервали или табулации. Така програма на Python няма как да работи коректно, ако не е удобна за четене. Началото на блок се означава с двоеточие, самият блок е отместен навътре спрямо обхващащия го, а краят на блок се означава просто като отместването се намали с една позиция. Липсата на отварящи и затварящи маркери поражда проблем с празни блокове. Можем да използваме ключовата дума pass, за да създадем празен блок.

Listing 6: Блокове

```
>>> if s1.startswith('aresto'):
    ...
    print("DA")
    ...
DA
```

Отместванията е допустимо да се извършват както с три интервала, така и с табулация, но не могат да се смесват. Кой вид отместване ще използваме зависи от предпочитанията ни и от хората, с които работим. Повечето разработчици на Python използват интервали, защото в PEP (документацията за нововъведения в Python) 8 пише така. PEP 8 всъщност никога не е бил предназначен за стандарт извън Python проекта, но се е превърнал в такъв.

Условни оператори

Условните оператори дават възможност изпълнението на програмата да се разклонява в зависимост от някакво условие. Условието е израз, чиято стойност може да се интерпретира като True или False. Това може да са литералите True и False (с главни първи букви), променливи, логически изрази или структури.

Условни оператори

```
1  if password == "123456":
2      print("Welcome!")
3   else:
4      print("Invalid password"
5   num = input('Enter a number: ')
6   if num > 0:
7      print(num, "The number is positive")
8   elif num < 0:
9      print("The number is negative")
10  else:
11     print('The number is zero')
```

Условни оператори

Условните оператори могат да се влагат един в друг. Една от най-честите употреби на условния оператор е когато се извършва сравнение на две стойности. Тези стойности могат да бъдат както на литерали, така и на променливи или изрази.

Listing 7: Вложени условни оператори

```
1 if attack_detected:
2     if defcon < 3:
3         doomsday_device.kaboom()
```

Таблица: Оператори за сравнение

Оператор	Значение
x == y	x е равно на y
x < y	x е по - малко от y
x > y	x е по - голямо от y
x <= y	x е по - малко или равно на y
x >= y	x е по - голямо или равно на y
x != y	x не е равно на y
x is y	x и y са един и същ обект
x is not y	x и y са различни обекти
x in y	x е елемент на контейнера y
x not in y	x не е елемент на контейнера y

Булеви оператори

В Python има различни неща, които могат да бъдат интерпретирани в булев контекст.

Таблица: Стойности, интерпретирани като булева неистина

Оператор	Значение
False	Литералът, изразяващ булева неистина
None	Ключова дума, означаваща “нищо”
0	Стойността нула
””	Празен символен низ
()	Празен кортеж
[]	Празен списък
{}	Празен речник

Таблица: Стойности, интерпретирани като булева истина

Оператор	Значение
True	Литералът, изразяващ булева истина
1	Всяка числова стойност, различна от 0
2	Всяка числова стойност, различна от 0
'qq'	Всеки непразен символен низ
(1,2)	Всеки непразен кортеж
[1, 1]	Всеки непразен списък
{a:1, b:2}	Всеки непразен речник

Булеви оператори

Булевите оператори имат интересното свойство да проверяват само това, което е необходимо. Например x and y очаква И x И y да са верни, така че е достатъчно x да не е вярно, за да бъде целият израз False. Затова когато Python тръгне да оценява този израз, ако x е False, то той не проверява y. Всъщност ако x е False, се връща x, в противен случай се връща y. По аналогичен начин работи и операторът за логическо ИЛИ or. В израза x or y, ако x е вярно, то се връща x, в противен случай се връща y.

Булеви оператори

Ако върнатата стойност от input е истина, т . е . не е празен низ, то тогава тази стойност присвояваме на name, в противен случай (ако низът е празен) присвояваме стойността по подразбиране < unknown >.

Listing 8: Третиране на променливи в булев контекст

```
1 name = input("Enter your name") or '<unknown>'
```

Пример

За несемейни

Ако облагаемата сума е над	Но не повече от	Данъкът е	Върху сумата над
0	21 450	15%	0
21 450	51 900	3 217.50 + 28%	21 450
51 900	-	11 743.50 +31%	51 900

Таблица: данък несемейни

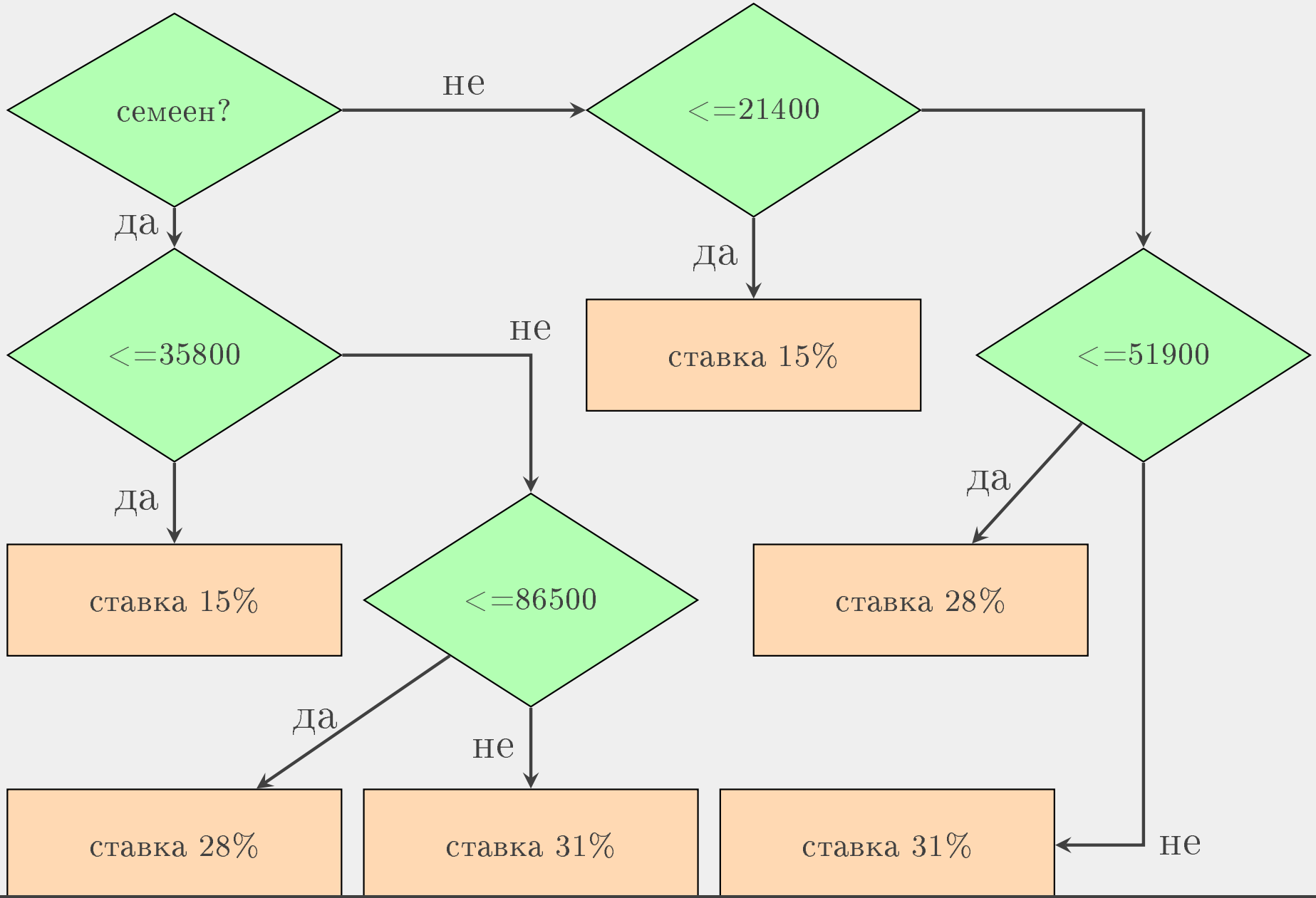
Пример

За семейни

Ако облагаемата сума е над	Но не повече от	Данъкът е	Върху сумата над
0	35 800	15%	0
35 800	86 500	5 370.00 + 28%	35 800
86 500	-	19 566.00 +31%	86 500

Таблица: данък семейни

Схема данък


семеен?
не
<=21400
да
да
<=35800
не
ставка 15%
<=51900
да
да
ставка 15%
<=86500
ставка 28%
да
не
ставка 28%
ставка 31%
ставка 31%
не

Многоклон

В някои езици има конструкция за многоклоново разклонение, в което израз, който не е булев, се оценява и сравнява с няколко предварително указани стойности. В езиците, в които такава конструкция не е налична, се използват вложени if-else конструкции със същия ефект.

Цикъл с предусловие

Listing 9: while

```
1 while condition:
2     block
```

Цикъл for

Съществуват и цикли, при които има управляваща променлива, чиято стойност се променя на всяка итерация по начин, указан в самата конструкция на цикъла, а не в тялото. Тъй като променливата често е от целочислен тип и се увеличава, цикълът се нарича също и цикъл с брояч.
Съществува разновидност на for цикъла, който итерира върху редица от елементи. В някои езици той се нарича foreach (за всеки), защото на всяка стъпка управляващата променлива е поредният елемент от редицата.

Цикъл for

Listing 10: for loop

```
1 a = ['John','Marie','George']
2 for name in a:
3     print(a)
```

Цикъл for

Тъй като итерирането върху числов интервал е често срещано в Python, има вградена функция за това – range. Функцията range работи като включва първият елемент, но изключва последния. Така в примера изведените числа ще са от 1 до 4. Ако бъде подаден само един параметър, то range генерира редица от 0 до последното цяло число преди стойността на параметъра. Функцията range има и незадължителен трети параметър, указващ стъпката (положителна или отрицателна).

Listing 11: Функция range

```
1 for i in range(1,5):
2     print(i)
```

Break

Операторът break е предназначен за прекъсване на изпълнението на цикли в произволно място в тялото на цикъла. Изпълнението на оператора break предизвиква излизане от цикъла и предаване на управлението на оператора, записан непосредствено след цикъла. На практика break представлява безусловен преход към първия ред след края на цикъла.

Continue

Операторът continue е предназначен за прекъсване на изпълнението на текущата итерация на цикли. Изпълнението на оператора continue предизвиква преминаване към проверка на условието на цикъла.

Стил на писане. PEP8

Тип	Конвенция	Пример
функция	малки букви, разделяне с подчертавка	function, my_function
променлива	малки букви, буква, дума, думи	x, var, my_variable
клас	първа главна. Всяка следваща дума също, без "_"	Model, MyClass
метод	същото като функция	class_method, method
константа	само главни букви	CONT, MY_CONST
модул	къса дума или думи, с "_". Само малки букви	module.py, my_module
пакет	къса дума или думи. Само малки букви	package, mypackage

Стил на писане. PEP8

PEP 8 препоръчва максималната дължина на ред да бъде 79 символа.

Стил на писане. PEP8

Използването на празни редове за подобряване на четимостта:

- Функциите и класовете трябва да са разделени с два празни реда.
- Методите и дефинициите вътре в класове да са разделени с един празен ред.
- Вътре във функции отделни логически блокове могат да се разделят с празен ред.

За да изпълним програма на Python, просто записваме програмния код в текстов файл с разширение .py и го стартираме от средата за разработка, ако ползваме такава. В противен случай го стартираме от командния ред.
Стартирането от командния ред става като напишем името на интерпретатора (python или python3) и след него напишем името на файла, който съдържа нашия python код. Ако името на файла ни съдържа интервали, то то трябва да е оградено с кавички, защото в противен случай ще получавате съобщение за грешка от типа file not found.

Ако на първия ред сме указали пътя до интерпретатора
#!/usr/bin/python (за Linux),
то можем да направим програмата да се извиква като стандартен изпълним файл, като променим атрибутите ѝ.

Задача

Връзката между всеки две цели числа A и B може да бъде изразена по единствен начин при използване на други две цели числа – X и r както следва:
A = X пъти B + r

Задача

A div B показва колко пъти B се съдържа в A; A mod B показва какъв е остатъкът. Множеството на целите числа е затворено относно тези две операции. Ще изразим X и r от горния израз за целочислено деление на A и B така: X ←A div B r ←A mod B

Задача

Нека алгоритъмът работи в среда, състояща се от променливите A, B и r, които са от целочислен тип. Да изразим необходимите за алгоритъма на Евклид действия над средата. 1. r← A mod B 2. A ←B 3. B ←r Те спират да се повтарят когато се удовлетвори условието r=0.

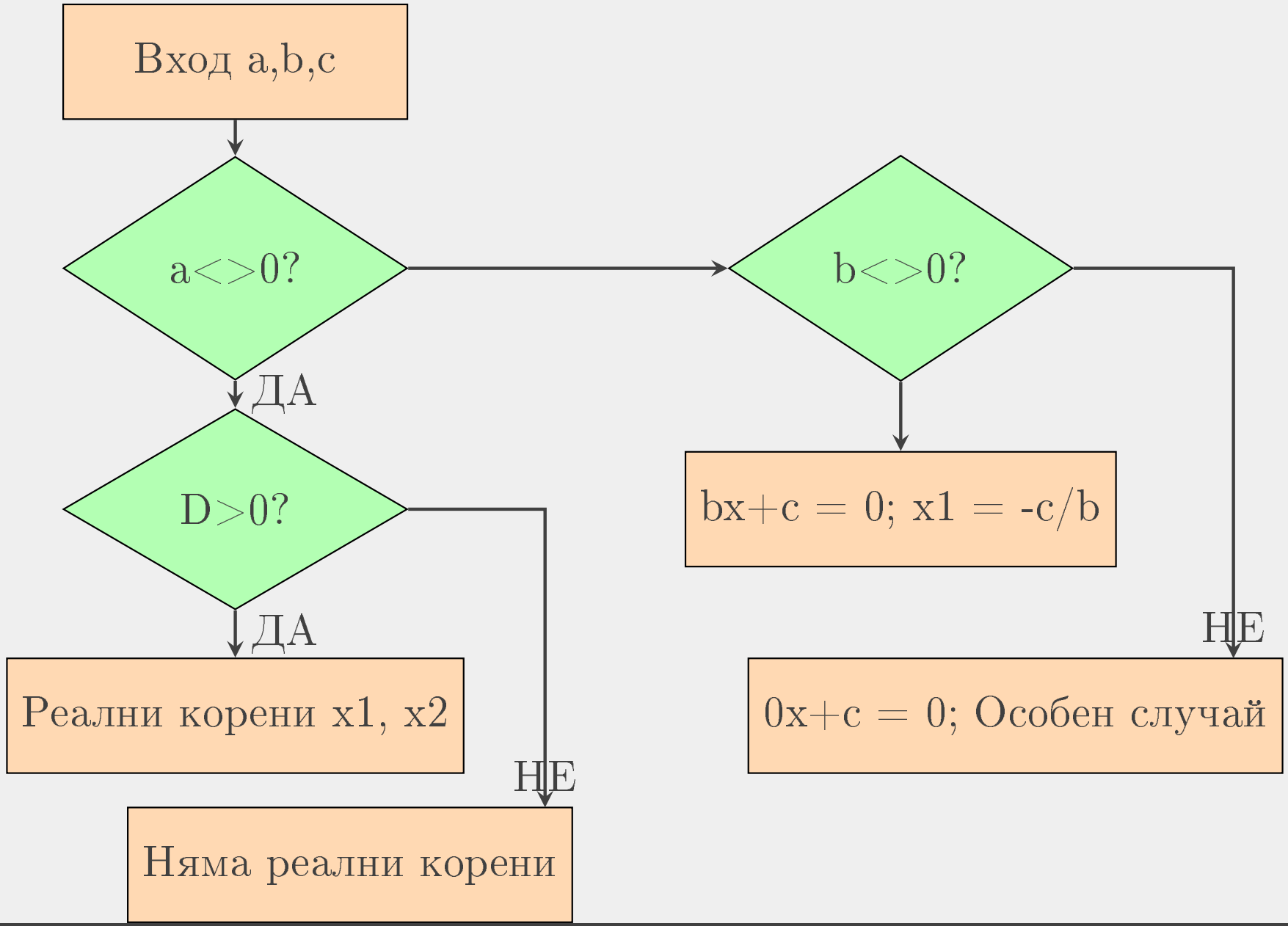
Задача

Реализирайте алгоритъма на Евклид като Python програма, която приема от входа две числа и извежда най-големия им общ делител.

Задачи

Да се напише програма, която намира корените на квадратно уравнение, ако има такива.

Задача


Вход a,b,c
a<>0?
ДА
D>0?
ДА
Реални корени x1, x2
НЕ
Няма реални корени
b<>0?
bx+c = 0; x1 = -c/b
НЕ
0x+c = 0; Особен случай

Благодаря за вниманието!

Въпроси?