

РЪКОВОДСТВО за Python

Кратко обяснение на основните концепции за кодиране

от Windmaran Network

Ниво 1:

Присвоения, променливи, изрази, константи,
аритметични оператори

Ниво 2:

Ако оператори, if-else оператори,
оператори за сравнение, логически оператори

Ниво-3:

Цикъли, списъци, функции

Copyright © 2010-2021 Windmaran Network. Всички права са запазени. Никакви части от този документ, дизайн на корицата и икони не могат да бъдат възпроизвеждани или превеждани под каквато и да е форма, по какъвто и да е начин (електронно, фотокопиране, запис или по друг начин) без предварителното писмено разрешение на издателя.

Връзка: www.windmaran.com/wm/pdf/python-manual.pdf

Това издание публикува през май 2021
г. Windmaran Network
www.windmaran.com



Ръководство за Python от Windmaran Network

Ниво 1	5-22	Присвоения, променливи, изрази, константи, аритметични оператори
Ниво 2	23-29	Ако оператори, if-else оператори, сравнение и логически оператори
Ниво 3	30-58	Цикъли, списъци, функции
Приложения	59-60	Приложение 1 Стартирайте програми на Python с помощта на Thonny
	61	Приложение-2 Стартирайте Python онлайн в браузъра

Относно този урок

Това е урок за програмния език Python. Предназначен е за студенти, които са завършили изучаването на Python Primer!

Това ръководство е разделено на три раздела (Ниво-1, Ниво-2 и Ниво-3) – което съответства на три нива на примери за кодиране, представени на уеб страницата на Windmaran Python. Покрити са следните понятия:

Ниво-1: Присвоения, променливи, изрази, константи, аритметични оператори

Ниво 2: if оператори, if-else оператори, сравнение и логически оператори

Ниво 3: цикли, функции, списъци

Текстът е разделен на учебни модули. Всеки модул представя учебния материал в логичен, последователен ред.

Приложение 1 и 2

Приложение 1 представя Thonny - интегрирана среда за разработка на Python, която е предназначена за начинаещи

Приложение 2 обяснява как да стартирате Python код онлайн във вашия браузър

Следвайте ръководството за седемте стъпки

За да изучавате Python по подреден начин, е необходимо да следвате инструкциите, описани в Ръководството за седемте стъпки. Започнете от стъпка № 4:

Връзка: www.windmaran.com/wm/wm-aux-pages/seven-steps.php

Други ръководства за Python

Кратко ръководство за Python

Кратък преглед на функциите на Python:

Връзка: www.windmaran.com/wm/pdf/python-quick-reference.pdf

Препоръчителни връзки за изучаване на Python

W3schools: W3schools.com е сред най-добрите онлайн уроци за изучаване на Python
връзка: <https://www.w3schools.com/python/>

tutorialspoint: урок за Python начинаещи връзка:
<https://www.tutorialspoint.com/python/index.htm>

Python за деца: игрово въведение в програмирането връзка:
<https://epdf.pub/python-for-kids-a-playful-introduction-to-programming.html>

Официалният дом на връзката на езика за програмиране Python:
<https://www.python.org/>

Научете се да кодирате с Thonny — IDE на Python за начинаещи
<https://thonny.org/>

Научете се да кодирате на python на
[repl.it https://repl.it/talk/learn/Learn-To-Code-In-Python/7484](https://repl.it/talk/learn/Learn-To-Code-In-Python/7484)

Научете Python онлайн: уроци по Python за разработчици от всички нива на
умения <https://realpython.com>

Практичен урок по Python 3 от д-р Андрю Н. Харингтън <http://anh.cs.luc.edu/python/hands-on/3.1/>

Ниво 1

Модул-1

Имена на променливи, аритметични изрази, присвояване, модулен оператор, функция `print()`.

Променливи в Python

Да поговорим за променливите. Името на променливата в Python може да бъде една буква или описателно име като: `accountNumber`, `totalAmount` и т.н.

За даване на имена на променливи се използва така нареченият „camelCase style“. CamelCase е конвенция за именуване, при която всяка дума в съставната дума се изписва с главни букви, с изключение на първата дума. Разработчиците на софтуер често използват camelCase, когато пишат изходен код. Примери за имена на променливи, които следват тази конвенция: `keepGoing` `acctNumber` `totalAmt` `creditCardNumber` `poštanskiCode`

Правила на Python за именуване на

променливи: • Имената на променливите не могат

да съдържат интервали. • Променлива започва с буква от A до Z или от a до z или долна черта (`_`), последвана от нула или повече букви, долни черти и цифри (0 до 9)

• Python не позволява препинателни знаци като `@`, `$` и `%` вътре
променливи •

Имената на променливите са чувствителни към главни букви.

Създаване на променливи

За разлика от други езици за програмиране, Python няма команда за деклариране на променлива. Променлива се създава в момента, в който за първи път ѝ присвоите стойност:

```
a = 100
```

Това означава: променлива (a) е създадена и е присвоена стойност 100.

Край на изявленията

За да завършите изявление в Python, не е нужно да въвеждате точка и запетая или друг специален знак; просто натиснете Enter.

Точка и запетая като разделител на изрази

Python позволява повече от един израз на един ред. Точка и запетая ';' се използва за разделяне на множество изрази на един и същи ред. Например, три изявления могат да бъдат написани като:

```
a = 1; b = 2; печат(a+b)
```

За интерпретатора на Python това би било идентично с този набор от изрази:

```
a = 1 b
= 2
печат(a+b)
```

Многоредови изявления

Не можете да разделите израз на няколко реда в Python, като натиснете Enter. Вместо това използвайте обратната наклонена черта (\), за да посочите, че даден израз продължава на следващия ред. Например това изявление:

```
a = 5 + 6 + 7
```

може да се пренапише по следния начин: (обърнете внимание на обратната наклонена черта в края на редовете):

```
a = 5
    + 6      \ \
    + 7
```

Функция print()

Функцията print() отпечатва определеното съобщение на екрана. Съобщението може да бъде низ или всяка друга променлива, променливата ще бъде преобразувана в низ, преди да бъде записана на екрана.

Нека разгледаме простото използване на функцията `print()`. Имайте предвид, че низовете са затворени в двойни кавички:

Пример 1:

```
печат („Програмата започва“)
a = 4
печат(a)

b = 2
печат(b)

print("Краят")
```

Програмата ще изведе:

```
Програмата започва
4 2
```

Край

Използвайте запетая за разделяне на променливи: ако искаме да отпечатаме комбинация от няколко променливи и текстов низ, трябва да разделим всеки елемент със знак за запетая.

Пример 2:

```
печат („Програмата започва“)
a = 4
b = 2
печат ("стойност на a:", a, "стойност на b:", b);

print("Краят")
```

Програмата ще изведе:

```
Програмата започва
стойност на a: 4 стойност на b: 2
Край
```

Как да печатате без прекъсване на ред

В Python можем да използваме параметъра `'end='` във функцията за печат. По подразбиране крайният параметър използва `'\n'`, което създава новия ред, но можем да зададем различна стойност за крайния параметър. Това му казва да завърши низа със символ по наш избор, вместо да завършва с нов ред.

Нека сравним следващите два примера:

Този пример използва версия по подразбиране на функцията `print()`, която автоматично добавя символ за нов ред, като по този начин прекъсва изходния ред.

Пример 3:

```
a = 1000
b = 2000 c
= 3000
печат (a)
печат (b)
печат (c)
```

Програмата ще изведе: 1000

```
2000
3000
```

Обратно, този пример завършва всеки ред със знак запетая (с изключение на последния израз на `print()`):

Пример 4:

```
a = 1000
b = 2000 c
= 3000
печат (a, край = ",")
печат (b, край = ",")
печат (c)
```

Програмата ще изведе:

```
1000,2000,3000
```

Аритметичен оператор(и)

Python използва следните символи (оператори) в изчисленията:

```
+ е събиране - е
изваждане * е
умножение / е деление
```

Редът на операциите (наричан още правила за приоритет) определя как изразът трябва да бъде оценен. Умножението има по-висок приоритет от събирането. По този начин изразът $2 + 3 * 4$ се интерпретира като има стойност $2 + (3 * 4) = 14$

Така че правилото е, че освен ако не използваме скоби, умножението и деленето предхожда събирането и изваждането.

Прочетете внимателно следващите 2 примера, които демонстрират синтаксиса на аритметичните изрази на Python:

Пример 5:

```
a = 4
b = 2
x = (a + b) * (a - b)

print("Стойност на x: ", x)
```

Програмата ще изведе: Стойност на x: 12

Пример 6:

```
a = 4
b = a/2 c
= (a + b) * (a - b) d = a + b *
a - bx = c - d

print("Стойност на x: ", x)
```

Програмата ще изведе: Стойност на x: 2

Модулен оператор

Операторът по модул (%) връща остатък от деление. Например, когато разделим 10/3, остатъкът е 1. Когато разделим 10/5, остатъкът е 0. Както можете да видите, модулният оператор се използва, за да се намери дали дадено число се дели на друго число.

Пример 7:

```
a = 17
b = 5
c = a % b
print("Стойността на c е ", c)
```

Програмата ще изведе: Стойност на x: 2

Модул-2

Коментари, константи

Коментари на Python

Коментарите на Python се използват за обяснение на код, за да го направят по-четлив. В Python използваме хеш символа '#', за да започнем да пишем коментар.

Това е ред за коментар

Освен това коментарите могат да се поставят след края на изявлението, както показва следващият ред:

```
secDay = 24 * 3600 # Секунди в изчислението на деня
```

Още примери за коментари на Python:

```
# Това е доста добър пример # за това как  
можете да разпределите коментари # върху  
няколко реда в Python
```

Python константи

Често искаме да работим с променлива, която има фиксирана стойност - константа. Но не можете да декларирате променлива или стойност като константа в Python. Просто не го променяйте! Като се има предвид това - „дефиницията на константа“ може да изглежда така:

```
PI = 3,141592653589
```

Много ръководства за стил предлагат постоянни имена да се пишат с главни букви. Това е много добра практика, която се използва в това ръководство.

Други примери:

```
ЛУНА_РАДИУС = 1737,4           # в километри # km  
ЗЕМЕН_РАДИУС = 6378  
ЗЛАТЕН_РАДИУС = 1,61803399  
ОТНОШЕНИЕ_ФЕТ_ДО_МЕТРИ = 3,281
```

Модул-3

Булев тип данни, низове, конкатенация на низове

Булев тип данни

Boolean е тип данни в Python. Булевият тип данни може да бъде една от двете стойности, True или False. Някои езици използват true и false всички малки букви, но python използва варианта с главни букви, така че трябва да сте сигурни, че винаги изписвате тези думи с главни букви. Ние използваме булеви числа в програмирането, за да правим сравнения и да контролираме потока на програмата. (Повече за използването на булеви типове данни в Ниво-2.)

Пример 1

```
a = Вярно          # с главни букви b
= False # с главни букви

print("Стойност на a = ", a)
print("Стойност на b = ", b)
```

Изход:

```
Стойност на a = Вярно
Стойност на b = False
```

Струни

Низът на Python съхранява серия от знаци като "Лос Анджелис". Низът може да бъде всеки текст в двойни или единични кавички.

В нашето ръководство ще използваме изключително низове, оградени с двойни кавички:

```
myName = "Джонатан" myCity
= "Ню Йорк"
```

Конкатенация на низове

Процесът на комбиниране на низове се нарича "конкатенация". В Python комбинирате („свързвате“) низове с (+) оператор по следния начин:

```
a = "alpha" b =
"bet" # нека
конкатенираме a и bc = a + b
# стойността на c е "азбука"
```

Ето още един пример за конкатенация. Обърнете внимание на използването на единичен символен низ " ", използван за създаване на интервал между първото име и фамилията

```
firstName = "John" familyName
= "Harris" # стойността на
fullName ще бъде "John Harris" # (вижте допълнителното
пространство) fullName = firstName +
+ фамилия
```

Празен низ

Да предположим, че виждаме в някакъв код на Python това задание

```
txt = "" # задаване на празен низ
```

Да, това е как да дефинирате и присвоите празен низ

Как да зададете и изведете низове

Пример 2

```
град = "Лос Анджелис" щат =  
"Калифорния"  
  
# низът е разделен със запетая и единичен интервал myCity = city + ", " + state  
  
print("Живея в", myCity)
```

Пример 3

```
name1 = "Джон"  
name2 = "Дейвид"  
  
print("Моите братя са", name1, "and", name2)
```

Пример 4

```
txt = ""  
txt + "abc" txt # празен низ txt = txt =  
  
print("Стойността на txt е:", txt) # показва: "abcdef"
```

Добавяне на допълнително пространство

Може би вече сте забелязали, че Python добавя допълнително пространство при отпечатване на няколко параметъра

Пример 5

```
a = 10 b
= "alpha" c =
"epsilon" d = 20 #
допълнително
пространство, добавено към отделни изходни елементи: print
("a:",a,"b:",b,"c:",c,"d :",d)
```

Програмата ще изведе: a: 10 b: алфа c: епсилон d: 20

Модул-4

Кръгла функция, преобразуване на тип, преобразуване от число в низ, запълване на низове

Кръгла функция

Функцията round() връща число с плаваща запетая, което е закръглена версия на посоченото число с посочения брой десетични знаци.

Броят на десетичните знаци по подразбиране е 0, което означава, че функцията ще върне най-близкото цяло число. Например

```
x = round(5.76) # стойността на x ще бъде 6
```

Синтаксис:

кръг (число, цифри)

Пример 1:

```
pi = 3,14159265359
piRound = round(pi,2)
print("Стойността на pi е ", pi)
print("Стойността на piRound е ", piRound)
```

Програмата ще изведе:

```
Стойността на pi е 3.14159265359 Стойността
на piRound е 3.14
```

Пример 2:

```
a = 7,543543543
b = кръг(a) c =
кръг(a,1) d =
кръг(a,2) e =
кръг(a,3)

print("a=", a)
print("b=", b)
print("c=", c)
print("d=", d)
print("e=", e)
```

Програмата ще изведе:

```
a= 7,543543543
b= 8
c= 7,5
d= 7,54
e = 7,544
```

Преобразуване на числа и типове в Python

В Python има два числови типа (има и комплексни числа, които няма да обсъждаме):

int
float

int - те се наричат просто цели числа или цели числа, са положителни или отрицателни цели числа без десетична запетая.

float - наричат се float, представляват реални числа и се записват с десетична точка, разделяща целочислената и десетичната част.

пример:

```
x = -11          # цяло число
y = 3,14 # float
```

Можете да конвертирате от един тип в друг с функциите int() и float().

Важно: int(x)

ще преобразува (x) в цяло число и ще се отърве от десетичната част чрез съкращаване то (не закръгляване!). Например, int(2.95) ще се преобразува в цяло число 2

Примери: x =

```
1 # присвояване на int номер y = 2.8 #
```

```
присвояване на float номер
```

```
#конвертиране от int в float: a = float(x)
```

```
#конвертиране от float в int: b = int(y) c =
```

```
int(12.95) # 12.95 ще бъде преобразувано в
```

```
12 d = int(5/3) # 5/3 ще бъде преобразувано в 1
```

```
# конвертиране от int в float n = float(10) # ще
```

```
бъде преобразувано в 10.0 m = float(1) # ще бъде преобразувано в 1.0
```

Преобразуване на числа в низове

Python преобразува числата в низове чрез използване на функцията `str()`. Ще предадем число или променлива във функцията `str()` и след това тази числова стойност ще бъде преобразувана в стойност на низ.

Нека първо разгледаме преобразуването на цели числа. За да преобразувате цялото число 12 в низова стойност, просто предавате 12 във функцията `str()`:

```
str(12) # преобразува в низ "12"
```

Можем също да използваме функцията `str()` за преобразуване на числа с плаваща стойност или плаващи променливи в низ:

```
str(1/2) # преобразува в низ "0.5"
```

Pad String с интервали, използвайки функция `rjust()`

Функцията `rjust()` в python връща низа с допълване, направено в левия край на низа до определената дължина.

```
xx = "xyz" #
```

```
връщане на низ с дължина 10 знака (подпълнен със 7 водещи интервала) xx.rjust(10)
```

```
yy = 123 #
```

```
връщане на низ с дължина 5 знака (подплатен с 2 водещи интервала)
```

```
str(yy).rjust(5) # първо, преобразувайте yy в низ
```

Пример 1:

```
string = "моята котка"  
ширина = 9  
print("0123456789")  
print(string) # отпечатване  
на дясно подравнен низ  
print(string.rjust(width))
```

Програмата ще изведе:

```
0123456789  
моята котка  
      моята котка
```

Пример 2:

```
a = 6797,543543543  
b = 48,987487 c =  
7,387652  
  
aStr = str(round(a,2)) bStr =  
str(round(b,2)) cStr =  
str(round(c,2))  
  
print("aStr=", aStr)  
print("bStr=", bStr)  
print("cStr=", cStr)  
  
print("aStr=", aStr.rjust(8)) print("bStr=",  
bStr.rjust(8)) print("cStr=", cStr.rjust(8))
```

Програмата ще изведе (обърнете внимание на разликата между преди и след приложението rjust) следното:

aStr= 6797.54

```
bStr = 48,99  
cStr = 7,39  
aStr = 6797,54  
bStr=      48,99  
cStr=      7.39
```


Модул-5

Изявление за импортиране, произволни функции: произволно цяло число, произволно плаващо число

Изявлението за внос

Python получава достъп до кода в друг модул чрез процеса на импортирането му. Във файл на Python това ще бъде декларирано в горната част на кода. Например Python има вграден модул, който можете да използвате, за да правите произволни числа. За да получите достъп до произволна функция, декларирайте това изявление

импортиране на случаен принцип

Случайна целочисленна функция

Python randint() е функция на произволния модул в Python. Случайният модул дава достъп до различни полезни функции, като една от тях може да генерира произволни числа.

Синтаксис: random.randint(начало, край)

Функцията връща произволно цяло число в дадения диапазон като параметри. Обърнете внимание на изявлението „импорт на случаен принцип“ в горната част.

Пример 1:

```
импортиране на случаен принцип
i1 = random.randint(0,1) i2 =
random.randint(1,100)
print("Стойност на i1:", i1)
print("Стойност на i2:", i2)
```

Програмата ще изведе:

Стойност на i1: 1

Стойност на i2: 48

Функция за произволно плаване

Функцията Python random() се използва за връщане на произволно число с плаваща запетая между диапазон <0,1> , 0 (включително) и 1 (изключително).

Синтаксис: random.random()

Следващият пример трябва да е достатъчен, за да обясни как работи функцията.

Пример 2

```
импортиране
на случаен принцип rand1 =
random.random() rand2 =
random.random() rand3 =
random.random()
print("rand1=", rand1)
print("rand2=", rand2) print("rand3=", rand3)
```

Изход (той е различен за всяко изпълнение на скрипт - по-долу е само едно от конкретните стартирания):

```
ранд1= 0,25173679172686325 ранд2=
0,9133861116499714 ранд3=
0,18862906774115207
```

Модул-6

Списък с математически функции

Ето математическите функции, които трябва да знаем:

Математическа функция	Действие	Примери
pow()	pow(x, y) връща стойността на x в степен y връща квадратния корен от x abs(x) връща абсолютната	pow(8, 2) # връща 64
math.sqrt()	(положителна) стойност на x math.ceil (x) връща стойността на	math.sqrt(25) # връща 5
корени на мускули()	x, закръглена нагоре до най-близкото цяло число math.floor() math.floor(x) връща стойността	abs (-9,7) # връща 9.7
math.ceil()	на x, закръглена надолу до най-близкото цяло число	math.ceil(4.991) # връща 5 math.ceil(10.5) # връща 11 math.ceil(4.001) # връща 5
		math.floor(4.01) # връща 4 math.floor(4.99) # връща 4

Следният пример показва математическите функции в действие (обърнете внимание на изрази „import math“ в горната част):

Пример 1

импортиране на математика

```
# pow() функция
print("pow()") res1 =
math.pow(2, 4) res2 =
math.pow(5, 3) res3 =
math.pow(10, 3) print("res1
= ", res1) print("res2 = ", res2)
print("res3 = ", res3) print()

# sqrt() функция
print("sqrt()") res1 =
math.sqrt(2) res2 =
math.sqrt(3) res3 =
math.sqrt(1000) print("res1 =
", res1) print("res2 = ", res2)
print("res3 = ", res3) print()

# ceil() функция
print("ceil()") res1 =
math.ceil(100.001) res2 =
math.ceil(-1.001) res3 =
math.ceil(-1.99) print("res1 = ",
res1) print("res2 = ", res2)
print("res3 = ", res3) print()

# abs() функция
print("abs()") res1 =
math.abs(-1.55) res2 =
math.abs(-987.123) res3 =
math.abs(1000.123) print("res1 =
", res1) print("res2 = ", res2)
print("res3 = ", res3) print()
```

Изход:

```
pow()
res1 = 16.0
res2 = 125.0
res3 = 1000.0

sqrt()
res1 = 1,4142135623730951 res2
= 1,7320508075688772 res3 =
31,622776601683793

ceil()
res1 = 101
res2 = -1 res3
= -1

abs()
res1 = 1,55
res2 = 987,123 res3
= 1000,123
```

Бележка относно терминологията (функция срещу метод)

В това ръководство говорихме за (rjust) или (randint) като функции. Но всъщност това са методи. Въпреки това, тъй като не се занимаваме с обектно-ориентирани функции, ще останем с тази доста "неточна" терминология. Имайте предвид това разграничение, в случай че ще изучавате обектно-ориентирани функции на Python.

Модул-7

Още примери за преглед

Покрихме много терен; сега е време да прегледаме това, което научихме. Прочетете внимателно тези примери:

Пример 1

```
a = 55
b = 7

c = a % b
d = int(a/b)
x = d * b + c

# стойността на x трябва да е равна на стойността
# на отпечатък ("x:", x, "c:", c, "d:", d)
```

Изход:

x: 55 b: 7 c: 6 d: 7

Пример 2

```
MARS_RADIUS = 3396.2 # в километри
MILES_RATIO = 1.609344 # една миля е равна на 1,609344 километра

print("Изчисляване на обиколката на Марс")

marsCirc = 2 * MARS_RADIUS * PI
print("Екваториалната обиколка на Марс е", str(round(marsCirc,3)), "km")
print("Екваториалната обиколка на Марс е", str(round(marsCirc/MILES_RATIO,3)), "мили")

print("Краят")
```

Изход:

Изчисляване на обиколката на Марс
 Екваториалната обиколка на Марс е 21338,954 км
 Екваториалната обиколка на Марс е 13259,411 мили
 Краят

Пример 3

```
cc = 1000 a
= cc/3,1234 b = cc/
47,754 c = cc/
278,96705

aStr = str(кръгла(a,3)) bStr =
str(кръг(b,3)) cStr = str(кръг(c,3))

print("aStr=", aStr.rjust(8)) print("bStr=",
bStr.rjust(8)) print("cStr=", cStr.rjust(8))
```

Изход:

```
aStr= 320,164 bStr=
20,941 3,585 cStr=
```

Горният пример може да бъде пренаписан чрез комбиниране на (str) и (rjust) функции в един израз:

Пример 4

```
cc = 1000 a
= cc/3,1234 b = cc/
47,754 c = cc/
278,96705

# формат чрез закръгляване до 3 знака след десетичната запетая и
подравняване надясно print("a=", str(round(a,3)).rjust(8)) print("b=",
str(round(b,3)).rjust(8)) print("c=", str(round(c,3)).rjust(8))
```

Изход: a=

```
320,164 b=
20,941 3,585 c=
```

Това е краят на НИВО-1; Следващите две дейности са следните:

а) Разгледайте примерите на Python ниво 1 (група А и В)
Връзка: www.windmaran.com/wm/python/py-main.php?px=L1A1

б) Отговорете на въпросите от теста за онлайн ниво 1 на
Python Връзка: www.windmaran.com/wm/python/py-quiz.php?px=L1Q1

Ниво 2

Модул-8

Инструкция If, отстъп като маркер за блок

Вече знаем, че изразът `if` се използва за извършване на нещо въз основа на условието. Ако условието е вярно, тогава кодът с отстъп ще бъде обработен. Това означава, че Python разчита на отстъп (бяло пространство в началото на ред), за да дефинира обхвата в кода. В това ръководство ще използваме 4 интервала за отстъп.

След края на условието има задължително двоеточие, за да даде визуален индикатор, че кодов блок следва непосредствено след това.

За да демонстрираме значението на правилния отстъп в Python, нека сравним резултатите от два примера:

Пример 1:

```
a = 1
b = 1,
ако a == 0:
    a = a + 1 b
    = b + 1 x =
a + b
print("Стойност на x: ", x)
```

Програмата ще изведе: Стойност на x: 2

Нека направим малка „поправка“ – като премахнем отстъпа на твърдението: $b = b + 1$, което води до различен резултат!

Пример 2:

```
a = 1
b = 1,
ако a == 0:
    a = a + 1
b = b + 1
x = a + b
print("Стойност
на x: ", x)
```

Програмата ще изведе: Стойност на x: 2

За да обобщим правилата за отстъпа в Python:

а)

Python стриктно налага използването на отстъп за маркиране на блок от код: всички изрази със същото разстояние вдясно принадлежат на един и същ блок код. Ако блокът трябва да бъде по-дълбоко вложен, той просто е с отстъп надясно

б)

Обикновено за отстъп се използват четири бели интервала.

в)

Блокът завършва с първия ред без отстъп

г)

Двоеточие се използва за определяне на началото на кодов блок. С други думи, ролята на двоеточие в Python е да даде визуален индикатор, че кодов блок следва непосредствено след това.

Модул-9

Инструкция If – else, Инструкции if-elif-else

Инструкцията if-else изпълнява блок от код, ако определено условие е вярно. Ако условието е невярно, се изпълнява друг блок код. Нека разгледаме следния код:

Пример 1:

```
a = 2
b = 2
if a > b:
    a = a * 2
    b = b * 2
else:
    a = a + 1
    b = b + 1

x = a + b
print("Стойност на x: ", x)
```

Програмата ще изведе: Стойност на x: 6

Нека разширим нашето подценяване на синтаксиса if-else. Да предположим, че искаме да проверим за повече от едно условие. Синтаксисът за такъв случай е следният – обърнете внимание на новата конструкция „elif“:

ако условие 1:

код, който трябва да се изпълни, ако това условие е вярно; elif условие 2:

код, който трябва да се изпълни, ако първото условие е невярно и това условие е вярно;

друго:

код, който трябва да се изпълни, ако всички условия са фалшиви;

Ето една проста демонстрация (обърнете внимание на използването на задължителни двоеточия):

Пример 2

```
номер = -1 съобщ
=

ако числото > 0: msg
    = "Броят е положителен" elif number < 0:
msg = "Числото е отрицателно" else: msg =
    "Числото е 0"

печат(съобщ.)
```

Програмата ще изведе: Числото е отрицателно

Модул-10

Вложени оператори if, сравнение и логически оператори

Когато поставим оператор 'if' в друг оператор 'if' и евентуално повтаряме този процес няколко пъти, говорим за вложени.

Пример 1

```

a = 10 b
= 20 ако
b > a: ако a
    == b: a = a - 1 b
        = b - 1 друго:
        a = a + 1 b = b
    + 1

x = a + b
print("Стойност на x: ", x)

```

Python логически и оператори за сравнение

Нека си припомним сравнението и логическите оператори, които бяха описани в Python
грунд:

Изразяване	Python логически и оператор за сравнение
Се равнява	==
Не е равно	!=
По-голяма от	>
По-малко от	<
По-голямо или равно на	>=
По-малко или равно на	<=
И	и
Или	или

Разгледайте следния пример:

Пример 1

```

a = 0 b
= a + 1

ако a > 0 или b > 0: a = a + 1
    b = b + 1

ако a > 0 и b > 0: a = a + 1 b =
    b + 1

x = a + b
print("Стойност на x: ", x)

```

Изход:

Стойност на x: 5

Предишният пример може да изглежда „труден“ – така че нека вземем предишния код на Python и вмъкнем оператор за проследяване, за да проучим как протича изпълнението:

Пример 3

```
a = 0
b = a + 1

ако a > 0 или b > 0: a = a + 1
    b = b + 1
    print("Trace1 a =", a, "b =", b)

ако a > 0 и b > 0: a = a + 1
    b = b + 1
    print("Trace2 a =", a, "b =", b)

x = a + b
print("Стойност на x: ", x)
```

Изход

Следа 1 a = 1 b = 2

Следа 2 a = 2 b = 3

Стойност на x: 5

Модул-11

Булеви променливи и условия If-Else

В раздела Ниво 1 научихме за булевия тип данни. Boolean може да има само две стойности, True или False (първата буква трябва да бъде главна). Нека проучим как да използваме булеви променливи, за да контролираме потока на програмата, използвайки условни изрази. В следващия пример трябва да е очевидно, че блокът след оператора 'if' ще бъде изпълнен, тъй като условието е вярно.

Пример 1

```
a = 0
b = а, ако
b: a = a + 1

x = a
print("Стойност на x: ", x)
```

Изход:

Стойност на x: 1

Нека леко модифицираме предишния код:

Пример 2

```
a = 0
b = a >= 0 # резултатът от това сравнение е вярно, ако b: a = a + 1

x = a
print("Стойност на x: ", x)
```

Изход:

Стойност на x: 1

Нека продължим с допълнителни модификации: забележете в следващия пример теста на две булеви стойности в оператора 'if'.

Пример 3

```
a = 10
n = 10
i = 0

b1 = a == n
b2 = a > n

ако b1 или b2:
    i = i + 1

x = i
print("Стойност на x: ", x)
```

Изход:

Стойност на x: 1

Това е краят на НИВО-2; Следващите две дейности са следните:

а) Разгледайте примерите на Python ниво 2 (група А и В)
Връзка: www.windmaran.com/wm/python/py-main.php?px=L2A1

б) Отговорете на въпроси от теста за онлайн ниво 2
на Python Връзка: www.windmaran.com/wm/python/py-quiz.php?px=L2Q1

Ниво 3

Модул-12

Докато цикъл

Цикълът while в Python се използва за итерация на блок от код, стига тестовият израз (условие) да е вярно. Синтаксисът е много подобен на оператора if - както се вижда по-долу.

```
while условие: #  
    изпълнява код, стига условието да е вярно
```

Операторът while е най-основният цикъл за изграждане в Python. Нека сравним две части от идентичен код, за да проучим поведението на "while loop":

Пример 1

```
n = 0,  
докато n < 2: n =  
    n + 1  
  
x = n  
print("Стойност на x: ", x)
```

Нека модифицираме горния код на Python, като добавим изявление за проследяване. Изходният списък показва как на променливите се присвояват стойности по време на всяка итерация:

Пример 2

```
n = 0,  
докато n < 2:  
    print("Track n: ", n) n = n + 1  
  
x = n  
print("Стойност на x: ", x)
```

Изход:

```
Писта n: 0  
Писта n: 1  
Стойност на x: 2
```

Следващият пример отпечатва всички нечетни числа между <1, 10>. Обърнете внимание, че изразът 'if', който определя дали дадено число е нечетно или четно (ако остатъкът от деленето на 2 не е нула, тогава по дефиниция числото е нечетно).

Пример 3

```
n = 1,
докато n <= 10: ако
    n%2 != 0: print(n,
        " е нечетно число") n = n + 1
```

Изход: 1

```
е нечетно число
3 е нечетно число 5 е нечетно
число
7 е нечетно число 9 е нечетно
число
```

Безкрайни цикли

Работата с цикли while представлява предизвикателство: грешка в кодирането може да причини безкраен цикъл. Безкрайният цикъл, както подсказва името, е цикъл, който ще продължи да работи завинаги. Ако случайно направите безкраен цикъл, това може да срина вашия интерпретатор на Python. Ето един много прост код, който ще работи "завинаги":

```
n = 1,
докато n > 0:
    печат(n)
```

друг екземпляр на безкрайния цикъл е представен по-долу. Тъй като променливата 'n' не се увеличава по време на всеки цикъл, условието while (n <= 10) ще бъде постоянно вярно!

```
n = 1
докато n <= 10:
    ако n%2 != 0:
        print(n, " е нечетно число") == липсва (n
        = n + 1) изявление тук ==
```

Ключова дума прекъсване

Ключовата дума break се използва за прекъсване на цикъла. В следващия пример цикълът while завършва, когато условието (i > 5) е вярно

```
# завършва цикъла, ако i е по-голямо от 5
i = 1
докато i < 10:
    print(i) ако i
    > 5:
        прекъсване
    i = i + 1
```

Модул-13

За цикъл

Цикълът `for` е по-сложен, но е и най-често използваният цикъл. Често циклите `for` използват „диапазон“, който е функция на Python – вижте примера:

Пример 1

```
n = 0
за i в диапазон (0, 2, 1): n = n + 10

x = n
print("Стойност на x: ", x)
```

Разглеждайки горния пример, може да си помислите, че (диапазон) е част от синтаксиса (`for loop`). Не е. `Range()` е функция на Python, която връща поредица от цели числа, което от своя страна позволява на цикъла `for` да превърта. Тъй като диапазонът създава списък с фиксирана дължина, цикълът `for` се изпълнява за фиксиран брой итерации.

Синтаксисът на функцията за диапазон е: диапазон(старт, стоп, стъпка)

начало е цяло число, указващо от коя позиция да започне стоп е цяло число, указващо на коя позиция да завърши. `step` е цяло число, определящо увеличението.

Много е важно да се отбележи, че последователността завършва преди достигане на стойността (стоп), с други думи, функцията `range()` генерира числа до, но не включително (стоп) номер

примери

диапазон (0, 5, 1) # генерира числова последователност: 0, 1, 2, 3, 4 диапазон (2, 9, 3) #
генерира числова последователност: 2, 5, 8

Освен това стойностите (начало) и (стъпка) не са задължителни. По подразбиране за (начало) е 0, а по подразбиране за (стъпка) е 1. Това означава, че следната дефиниция на диапазона:

```
диапазон(0,10,1)
може да се запише като:
range(10) # генерира числова последователност 0,1,2,3,4,5,6,7,8,9
```

В контекста на цикъла `for` тези две твърдения са идентични

```
за i в диапазон (0, 2, 1): за i в
диапазон (2):
```


Ето примери, които илюстрират поведението на циклите for:

Пример 2

```
# Извеждане на нечетни числа в интервал <0,9>

ограничение
= 9 за n в диапазон (лимит+1): ако
    n%2 != 0: print(n, " е нечетно
        число")
```

Изход: 1

```
е нечетно число 3 е нечетно
число
5 е нечетно число 7 е
нечетно число
9 е нечетно число
```

Нека модифицираме горния код - просто повторете, като използвате последователност от нечетни числа (това ще доведе до идентичен изход):

Пример 3

```
# Извеждане на нечетни числа в интервал <0,9>

за n в диапазон (1,10,2): print(n, " е
нечетно число")
```

Още една модификация показва, че последователността на диапазона може да бъде дефинирана отделно (произвеждайки идентичен изход):

Пример 4

```
# Извеждане на нечетни числа в интервал <0,9>

a = диапазон(1,10,2) за n
в a: print(n, " е нечетно
число")
```

Ключова дума Break, използвана в цикъла for

Вече научихме, че цикълът while може да бъде прекратен от ключовата дума break. Същата функционалност важи и за цикъла for, както показва следващият пример:

```
# завършва цикъла, ако i е по-голямо от 3: за i в
диапазон (100): ако i > 3:
```

```
    прекъсване
    печат(i)
```

while спрямо сравнение на цикъла for

Нека напишем и сравним две части от кода. Задачата е да се напише програма, която сумира числата от 1 до 100. И двете версии (докато и за) постигат един и същ резултат.

Пример 5

while цикъл	за цикъл
<pre>LIMIT = 100 sum = 0 i = 1 докато i <= LIMIT: sum = sum + i i = i + 1 печат("sum:", sum)</pre>	<pre>LIMIT = 100 sum = 0 за i в диапазон (1, LIMIT+1, 1): sum = sum + i print("sum:", sum)</pre>

Модул-14

Функции

Функцията на Python е блок от код, предназначен да изпълнява конкретна задача. Функцията се дефинира с ключовата дума (def), последвана от име ('maxnum' в нашия пример), последвано от скоби (). Двоеточие се използва за определяне на началото на кода на функцията.

Имената на функциите следват същата конвенция като имената на променливи.

Важно правило:

Уверете се, че вашата функция е декларирана, преди да е необходима (извикана). Имайте предвид, че тялото на функцията не се интерпретира, докато функцията не бъде изпълнена. Въпреки това, функциите могат да бъдат дефинирани в произволен ред, стига всички да са дефинирани, преди който и да е изпълним код да използва функция. Така че нека повторим:

Правилото: Всички функции трябва да бъдат дефинирани преди всеки код, който върши реална работа

Затова поставете всички твърдения, които работят, последни.

Пример 1

```
def maxnum(a, b):  
    r = a  
    if (r < b): r = b  
    върне r  
  
u = 10  
v = 20  
x = maxnum(u, v)  
print("Стойност на x: ", x)
```

Забележка за разликата между параметри и аргументи:

Когато говорим за функции, термините „параметри“ и „аргументи“ често се използват взаимозаменяемо, сякаш е едно и също нещо, но има много фина разлика:

а)

Параметрите са променливи, изброени като част от дефиницията на функцията.

б)

Аргументите са стойности, предавани на функцията, когато е извикана.

След като проучихме основите на циклите и функцията `for`, нека представим пример въз основа на тези концепции:

Пример 3

импортиране на случаен принцип

```
def най-голям номер(n1, n2, n3): най-
    голям брой = n1 ако (n2 > най-
    голям брой): най-голям брой = n2
    ако (n3 > най-голям брой):
    най-голям номер = n3 връща най-
    голям брой
```

```
print("Итерация за i          n1          n2          n3 най-голямо число")
в диапазон(5): # за
    всяка итерация генерирайте 3 произволни числа с плаваща
    стойност # в интервал <0, 1000> и намерете/отпечатайте най-голямото
    число a = random.random() * 1000 b = произволно. random() * 1000 c =
    random.random() * 1000 най-голям = най-голям брой(a, b, c)
```

```
aStr          = str(round(a,2)).rjust(10) =
bStr          str(round(b,2)).rjust(10) =
cStr          str(round(c,2)).rjust(10) najvećeStr =
str(round (най-голям,2)).rjust(10)
```

```
печат ("          ", (i+1), aStr, bStr, cStr, най-голям Str)
```

Изход:

Повторение	n1	n2	n3 най-голям брой
	683,15	133,59	870,13 870,13 380,46 931,9
12	54,36	931,9	
3	215,9	195,45	890,17 890,17
4	908,97	802,3	296,81 908,97
5	291.17	226,68	577,85 577,85

Модул-15

Обяснение на обхвата на променлива

Кодът на Python по-долу дефинира функцията 'maxnum'

```
def maxnum (a, b):  
    r = a; ако  
    r < b:  
        r = b;  
    върнете r
```

Имайте предвид, че функцията дефинира променлива r. Тази променлива съществува във функцията 'maxnum'. За други функции тази променлива е невидима. Техническият термин за тази характеристика се нарича обхват. Това означава, че променливата r има само локален обхват. Локален обхват означава: в обхвата на функцията 'maxnum'. Ако дефинираме променливи извън която и да е функция, тогава обхватът на такава променлива е глобален. Това означава: навсякъде в нашата програма глобална променлива може да бъде достъпна и модифицирана.

Много е важно да се разбере разликата между локални и глобални променливи. Затова нека повторим това отново:

Знаем, че Python дефинира променлива, използвайки оператор за присвояване:

```
x = 1 # това твърдение дефинира променлива x
```

Ако този израз е дефиниран в тялото на функцията, обхватът (или видимостта) на тази декларация/присвояване е ограничен до тялото на функцията. Въпреки това, ако този израз е поставен извън която и да е функция, тогава обхватът на дефинираните променливи е глобален. Използването на глобални променливи не е добра техника за кодиране, тъй като води до много проблеми. Опитен програмист може да използва при определени условия глобални променливи; Python определено позволява това. Но за нашите цели ще избягваме използването на глобални променливи!

Да повторя още веднъж: обхватът се отнася до видимостта на променливите. Това означава кои части от вашата програма могат да я виждат или използват. Техниката на кодиране, приложена в този курс, ограничава обхвата на променлива до една функция.

Как се прави това? Нека разгледаме следващия пример, който ще бъде шаблон за всички наши примери:

Програмен шаблон на Python def main():

```
    някакъв код тук.        . .
    # край на main()
```

```
def someFunction(): някакъв
    код тук. # край на        . .
    someFunction()
```

```
main()
```

Това, което виждате, е извикване на функция main (в самия край на програмната последователност), която предава изпълнението на кода към тялото на функцията main. С други думи, всички действия се случват вътре в тялото на функцията main. Други функции се извикват пряко или косвено от основната функция. Този стил гарантира, че всички променливи имат локален обхват - което е нашата цел. Това на пръв поглед ненужно усложнение от затварянето на целия код на Python в една или няколко функции гарантира надеждността на нашия код.

Не забравяйте, че сме установили правило за подреждане на функциите, посочени по следния начин:
 функциите могат да бъдат дефинирани в произволен ред, стига всички да са дефинирани, преди който и да е изпълним код да използва функция.

Представеният шаблон поставя изпълнимия код в един ред код - main(), който трябва да бъде поставен като последния израз! Нека демонстрираме това със следния пример:

Пример 1

```
# Програма на Python, която НЕ използва глобални променливи
def main(): a =
    10 b = 20
    x =
    maxnum(a, b)
    print("Стойност на x: ", x) # край на
    main()

def maxnum(a, b):
    r = a,
    ако r < b: r =
        b
    върнете r
    # край на maxnum()

# извикване на 'main' функция ТРЯБВА ДА Е ПОСЛЕДНИЯТ израз !!! main()
```

Забележка: Всички примери на уеб страницата на Python за Python – НИВО-3, Раздел-В използват тази техника.

Модул-16

Списъци

Какво е списък? Списъкът е специална променлива, която може да съдържа повече от една стойност наведнъж. Да кажем, че имаме списък с числа. Съхраняването им в единични променливи би изглеждало така:

```
число1 = 200
число2 = 300
число3 = 400
```

Но ако искаме да използваме цикъл, за да намерим конкретно число, това би било трудно – особено за голям набор от числа. Решението е да използвате списък. Списъкът може да съдържа много стойности под едно име и можете да получите достъп до стойностите, като се позовавате на индексен номер: `myList = [200, 300, 400]`

Основните характеристики на списъка са

- Списъкът може да съдържа много стойности под едно име •

Достъпът до елементите на списък се осъществява чрез техния индексен номер • Операторът за индексване `[]` избира един елемент от поредица. В

Изразът в квадратни скоби ~~се нарича индекс~~ и трябва да бъде цяло число.

- Списъчните индекси започват с 0. `[0]` е първият елемент от списъка, `[1]` е вторият, `[2]` е третият и т.н.

Забележка 1: Python (както повечето съвременни езици) използва така нареченото индексване на базата на нула. Това е изключително важно! Нека запомним и повторим: • `[0]` индекса на първия елемент

- `[1]` индекса на втория елемент

- ако дължината на списъка е дефинирана като `len`, тогава `[len-1]` е индексът на последния елемент

Забележка 2: неофициално списъците и масивите се третираат като идентични понятия. Въпреки това Python прави разлика между списъци и масиви (въпреки че те са много сходни). Въпреки това, ние няма да обсъждаме концепцията за Python масиви в това ръководство.

Създаване на списък

Синтаксис:

```
име на списък = [елемент1, артикул2, ...]
```

Пример :

```
числа = [200, 300, 400] # този списък има 3 елемента
```

Достъп до елементите на списък

Получавате достъп до елемент от списък, като се позовавате на номера на индекса. Този израз осъществява достъп до стойността на първия елемент в 'числа':

```
someVariable = numbers[0] # първият елемент
```

Функцията за дължина на списъка len()

Свойството дължина на списък връща броя на неговите елементи.

Пример:

```
прости числа = [2, 3, 5, 7] x =
len(прости числа) # дължината на простите числа е 4
```

Дефиниция на празен списък

Празен списък се дефинира по следния начин

```
myList = [] # дефинирайте празен списък x = len(myList) # дължината е
нула
```

Досега разгледахме основните неща на списъците на Python – време е да покажем примери. Първият пример показва основите:

Пример 1

```
numberSet = [200, 300, 400]

print("Първо число: ", numberSet[0]) print("Втори номер: ",
numberSet[1]) print("Последно число: ", numberSet[2])
```

Изход:

```
Първо число: 200
Второ число: 300
Последен номер: 400
```

Следващият пример дефинира и инициира списък. Следователно на елементите на списъка се присвояват произволни стойности:

Пример 2

```
импортиране на случаен принцип

numberSet = [0, 0, 0] setLen =
len(numberSet) за i в
диапазон(setLen): numberSet[i] =
random.randint(1,1000) print("Index:", i, "Value:", набор от
числа[i])
```


Изход:

Индекс: 0 Стойност: 276

Индекс: 1 Стойност: 822

Индекс: 2 Стойност: 562

Низови елементи в списъка

Списъкът може да съдържа низови елементи

Пример:

```
цветове = ["червено", "зелено", "оранжево", "синьо"] печат(цветове[0])  
# извежда 'червено' отпечатване(цветове[3]) # извежда 'синьо'
```

Булеви елементи в списъка

Ето пример за списък, който има 3 елемента, всички от които са зададени на False (с главни букви):

```
boolVector = [False, False, False]
```

Модул-17

Списък с функции

Добавете нов елемент към списък – append():

Функция append добавя елемент в края на списъка

```
fruits = [] # нека започнем с празен списък fruits.append("Apple") # задава fruits[0] на "Apple"  
fruits.append("Mango") # задава плодове[1] на "Mango"
```

Трябва да е очевидно, че след изпълнение на 'append' дължината на списъка се увеличава с 1.

Изтриване на последния елемент от списъка – pop()

Функция pop премахва последния елемент от списък; дължината на списъка намалява с 1

```
fruits = ["Orange", "Lemon", "Avocado"] fruits.pop() #  
Премахва последния елемент ("Avocado") от списъка
```

Изтрийте елемент от списък по индекс с помощта на оператор `del`

Операторът `del` премахва елемент от посоченото местоположение на индекса от списъка.
пример:

```
стойности = [100, 200, 300, 400, 500, 600]
# Използвайте del, за да премахнете по индекс
del values[2] print(values)
```

Вмъкнете елемент на посочената позиция, като използвате функцията `вмъкване`

Функцията `insert()` вмъква елемент в посочения индекс.

Синтаксис:

```
listName.insert(index, element)
```

пример:

```
стойности = [100, 200, 300, 400, 500, 600]

values.insert(2,999) # вмъкнете 999 в позиции стойности[2] print(values)
values.insert(0,777) # вмъкнете 777 в стойности на позиции[0] print(values)
```

Очевидно е, че всяко вмъкване увеличава дължината на списъка с 1.

Премахнете всички елементи от списък – функция `clear()`.

Функцията `clear()` премахва всички елементи от списък.

Синтаксис

```
listName.clear()
```

пример:

```
метали = ["желязо", "мед", "сребро", "цинк"]

metals.clear() # премахва всички елементи – len(metals) връща 0
```

Следният пример показва всички списъчни функции в действие:

Пример 1

```
цветове = [] # започнете с празен списък # добавете 4  
елемента colors.append("red") colors.append("black")  
colors.append("white") colors.append("blue") print("trace  
-1", цветовете)  
  
# изтриване на последния елемент  
colors.pop() print("trace-2", цветовете)  
  
# вмъкване на 3-та и 1-ва позиция (индекси като [2] и [0]) colors.insert(2,"green")  
print("trace-3", colors) colors.insert(0, "yellow") print("следа-4", цветовете)  
  
# изтриване на втория (индексът е [1]) елемент del colors[1]  
print("trace-5", цветовете)
```

Изход:

```
следа-1 ['червено', 'черно', 'бяло', 'синьо'] следа-2 ['червено',  
'черно', 'бяло'] следа-3 ['червено', 'черно', 'зелено', 'бяло']  
следа-4 ['жълто', 'червено', 'черно', 'зелено', 'бяло'] следа-5  
['жълто', 'черно', 'зелено', 'бяло']
```

Модул-18

Елементи на списъци с цикъл, списък за копиране, списък за печат

Елементи на списък с цикъл

Да предположим, че трябва да построите цикъл, който ще итерира всички елементи на списъка. Знаем как да го направим с помощта на функцията `range()`:

Пример 1

```
числа = [100, 200, 300] дължина =  
len(числа) за i в диапазон  
(дължина): xx = числа[i] печат(xx)
```

Изход:

```
100  
200  
300
```

Въпреки това, Python позволява да преминете през всички елементи на списък. Предишният пример може да бъде пренаписан (и опростен) по следния начин (произвеждане на идентичен изход):

Пример 2

```
числа = [100, 200, 300] за xx в  
числа: print(xx)
```

Как да направите копие на списъка

Не можете да създадете копие чрез следното присвояване `a = [100, 200, 300]` `b = a` # това присвояване не прави копие на

Направете копие на списък с метода `list()`:

Пример 3

```
numberSet1 = [101, 102, 103, 104]  
numberSet2 = list(numberSet1)  
print(numberSet2)
```

Печатайте списъци по различни начини

Нека разгледаме начините за отпечатване на списък:

а)

вече видяхме най-простия възможен начин: `a = [1, 2, 3, 4, 5]`
`print(a)`

б)

отпечатайте списък с помощта
на цикъл `a = [1, 2, 3, 4, 5]` за `y`
в `a`: `print(y)`

в)

отпечатайте списък с помощта на цикъл и направете изхода на един ред (без прекъсвания на ред):
`a = [1, 2, 3, 4, 5]` за `y` в `a`: `print(y, end=" ")`

Модул-19

Разни теми: сравняване на плаващи числа, предаване на променлива и списък
на функция

Сравнение на числа с плаваща запетая

Поради грешки при закръгляването повечето числа с плаваща запетая се оказват леко неточни.
Това означава, че числата, които се очаква да бъдат равни, често се различават леко и обикновеният тест
за равенство се проваля. Следващият пример илюстрира проблема:

Пример 1

```
a = 0,15 + 0,15 b = 0,1
+ 0,2

if a == b:
    print("Стойностите a и b са равни") else:
    print("Стойностите a и b НЕ са равни")
```

Изход:

Стойностите a и b НЕ са равни

Това не е резултатът, който искаме да видим. Нека модифицираме горния пример, като въведем
персонализирана функция, наречена `nearllyEqual`

Пример 2

```
def nearlyEqual(x, y, epsilon): ret = False,
    ако abs(ab) < epsilon: ret = Вярно

    връщане рет

епсилон = 0,001; a =
0,15 + 0,15 b = 0,1 +
0,2

if nearlyEqual(a, b, epsilon):
    print("Стойностите a и b са равни") else:
    print("Стойностите a и b НЕ са равни")
```

Изход:

Стойностите и b са равни

Трябва да се подчертае, че използването на абсолютната разлика (наречена епсилон) не е наистина валиден метод. Но е добре да го използвате в прости ситуации. По-пълният отговор е сложен и е извън нивото на този документ. За повече подробности, моля, вижте тази статия:

<https://floating-point-gui.de/errors/comparison/>

Предаване на променлива във функция

Ако предадете променлива на функция, нейната стойност се копира и предава на функцията (преминава по стойност). Следователно функцията не може да промени променливата. Ето примера за функция, която демонстрира как работи тази функция:

Пример 3

```
def calcNumbers(m, n): m =
    m + 1 n = n + 1

    връщане (m + n)

a = 10
b = 20

x = calcNumbers(a, b) #
стойностите на a, b не се променят във функцията calcNumbers()
print("Стойност на a:", a) print("Стойност на b:", b) print("Стойност на x :", x)
```

Изход:

```
Стойност на a: 10
Стойност на b: 20
Стойност на x: 32
```

Прехвърляне на списък към функция

Предаването на списък към функция е различно – функцията може да променя/изтрива/добавя елементите на списъка. Разгледайте този пример:

Пример 4

```
def processList (списък1, списък2): #
    модифицира списък1
    list1.append(40) за i в
    диапазон(len(list1)): list1[i] = list1[i]
        + 1

    # модифициране
    на list2 list2.append(100)
    list2.append(200)
    list2.append(300)

числа = [10, 20, 30] моя
списък = []

processList (числа, myList)
print("numbers:", numbers)
print("myList:", myList)
```

Изход:

```
числа: [11, 21, 31, 41] myList:
[100, 200, 300]
```

Модул-20

Преглед на примери

Този модул служи като обобщение на това, което научихме за циклите, списъците и функциите. Най-добрият начин да проверим това, което сме научили в нашия курс по Python, е внимателно да прегледаме следващите примери. Моля, проучете кода, не се колебайте да експериментирате, като добавите изходни отчети за проследяване.

А. Изберете произволно числа от списък

Пример 1

импортиране на случаен принцип

БРОЙ = 5

числа = [12, 78, 458, 800, 2965, -568, 23450, -2, -1777] print("числа:", числа)

print("Случайно избрани числа:") за i в диапазон(БРОЙ):

randIdx = random.randint(0, COUNT-1) print("index:",
randIdx, "number:", numbers[randIdx])

Изход:

числа: [12, 78, 458, 800, 2965, -568, 23450, -2, -1777]

Случайно избрани числа: индекс: 0 номер: 12

индекс: 3 номер: 800 индекс: 4 номер: 2965

индекс: 2 номер: 458

индекс: 1 номер: 78

Б. Намерете най-малкия и най-големия елемент

В следващия пример ще създадем и попълним списък с произволни числа и след това ще намерим най-малкия и най-големия елемент:

Пример 2

```
импортиране на случаен принцип

SET_LEN = 50
числаНабор = []

# задайте елементите на списъка на произволни числа <0, 1000>
за i в диапазон(SET_LEN):
    numberSet.append(random.random() * 1000)

# намиране/присвояване на мин. и максимални
стойности minNum = numberSet[0] maxNum =
numberSet[0]

за i в диапазон (SET_LEN):
    ако minNum > numberSet[i]: minNum
        = numberSet[i] ако maxNum <
        numberSet[i]: maxNum = numberSet[i]

print("минималната стойност е: ", round(minNum,3))
print("максималната стойност е: ", round(maxNum,3))
```

Изход:

минималната стойност е: 8,186

максималната стойност е: 975,476

Този процес се повтаря, докато средният елемент е равен на търсения елемент или ако алгоритъмът установи, че търсеният елемент изобщо не е в дадения списък.

Следващият пример показва двоичното търсене в действие – обърнете внимание на наличието на проследяващ оператор за печат, който показва промените в диапазона от търсени елементи.

Пример 1

импортиране на математика

```
# Числа за двоично
търсене = [11, 27, 32, 38, 42, 64, 87, 91] # цел на списък с възходящ ред = 87
```

```
print("Двоично търсене в списък, сортиран във възходящ ред")
print("numbers=", numbers) print("target=", target)
```

```
най-малък = 0
най-голям = len(числа) - 1 намерен
            = Грешно
            = 1 броене
```

```
print("брой най-малката най-голяма средна средна стойност")
```

```
докато най-малкият <= най-голям и намерен == False:
```

```
    среден = math.floor((най-малък + най-голям) / 2) midValue =
    числа[среден] s1 = str(count).rjust(5) s2 = str(най-малък).rjust(8)
    s3 = str(най-голям).rjust(7) s4 = str(middle).rjust(6) s5 =
    str(midValue).rjust(8) print(s1, s2, s3, s4, s5) ако целта == средна
    стойност: # Да, целта е намерена print("целта е намерена")
    found = Вярно друго: ако целта < midValue: # целта е по-малка
    от средния елемент най-голям = среден - 1 else: # целта е по-
    висока от средния елемент, най-малък = среден + 1 брой =
    брой + 1
```

```
ако е намерен == False:
```

```
    print("целта НЕ е намерена")
```

Изход:

Двоично търсене в списък, сортиран във възходящ ред числа=
[11, 27, 32, 38, 42, 64, 87, 91] цел= 87 брой най-малък най-голям
среден средна стойност 3 38 5 64 6 87

1 7 2 7 3 7 целта е
намерена 4
6

Имайте предвид, че тази конкретна реализация на двоично търсене работи само във възходящ ред. Проверете реализацията в скрипта на Python-Level-3 "Бинарно търсене", който обработва и двете - възходящ и низходящ ред на елементите.

Модул-22

Сортиране с балончета

Сортирането с мехурчета получава името си, защото елементите са склонни да се движат нагоре в правилния ред като мехурчета, издигащи се на повърхността. Този прост алгоритъм се представя лошо в реалния свят и се използва предимно като образователен инструмент.

Обяснение на алгоритъм за сортиране с балончета: Сортирането с балончета е алгоритъм за сортиране, който работи чрез многократно преминаване през списък, който трябва да бъде сортиран, сравняване на всяка двойка съседни елементи и размяната им, ако са в грешен ред. Тази процедура на преминаване се повтаря, докато не се изискват размяна, което показва, че списъкът е сортиран.

Пример 1

```
# Номера за
сортиране на балон = [876, 356, 700, 247, 111]

print("Сортиране с
балончета") print("числа:", числа)

броя = 0
разменен = Вярно,
докато е разменен:
    разменен = False # обратно i = 1
    докато i < len(числа):

        # проверете дали двойката не е в ред, ако
        числа[i-1] > числа[i]:

            # разменете двойката с помощта на
            numbers[i-1] temp = numbers[i]
            numbers[i] = temp swapped = True # това ще
            принуди да повторите цикъла 'while' i = i + 1

    ако е
    разменен: count =
    count + 1 print("count=", count, end=" ")
    print("numbers:", numbers) else:
    print("списъкът е сортиран - сортирането с
    балончета завършва")

print("Краят")
```

Изход:

Сортиране с
балончета: [876, 356, 700, 247, 111] брой = 1 числа:
[356, 700, 247, 111, 876] брой = 2 числа: [356, 247, 111, 600], брой =
87 числа: [247, 111, 356, 700, 876] count= 4 числа: [111, 247, 356, 700,
876] списъкът е сортиран - сортирането с балончета завършва

Край

Разгледайте сортирането с балончета, реализирано като функция в скрипт Python-Level-3 Script "Bubble Вид"

Модул-23

Сортиране по избор

Сортирането по избор е един от най-простите алгоритми за сортиране. Работи по следния начин: • Намерете най-малкия елемент. Разменете го с първия елемент. • Намерете втория най-малък елемент. Разменете го с втория елемент. • Намерете третия най-малък елемент. Разменете го с третия елемент. • Повтаряйте намирането на следващия най-малък елемент и го разменяте в съответната правилна позиция, докато списъкът бъде сортиран.

Ето една практическа демонстрация. Имайте предвид, че броят на преминаването на сортирането през списъка е с един по-малко от броя на елементите в списъка.

```
myList = [64, 25, 12, 22, 11]
```

```
# Намерете минималния елемент в myList [0...4] # и го поставете  
в началото 11 25 12 22 64
```

```
# Намерете минималния елемент в myList [1...4] # и го поставете  
в началото на myList [1...4] 11 12 25 22 64
```

```
# Намерете минималния елемент в myList [2...4] # и го поставете  
в началото на myList [2...4] 11 12 22 25 64
```

```
# Намерете минималния елемент в myList [3...4] # и го поставете  
в началото на myList [3...4] 11 12 22 25 64
```

Разгледайте сортирането по избор, реализирано като функция в скрипт Python-Level-3 Script "Избор Вид"

Пример 1

```

# Избор Сортиране
на номера = [876, 356, 700, 247, 111]

print("Сортиране по избор")
print("числа:", числа)

# намиране на минимален елемент (numbersLen-1) пъти = 1
диапазон(брой, minIdx, j0) за i в
    range(1, numbersLen - брой)

    print("count=", count, "minIdx=", minIdx, "j0=", j0)

    # тест срещу елементи, започващи j0, за да се намери най-малкият j = j0,
    докато j < numbersLen: # ако този елемент е по-малък, тогава това е
    новият минимум if numbers[j] < numbers[minIdx]: minIdx = j # намерен нов
    минимум - актуализира неговия индекс j = j + 1

    if minIdx != i: #
        разменете елементите - избягване на размяната на елемент със себе си =
        = temp                numbers[i] temp numbers[i] = numbers[minIdx] numbers[minIdx]

    print("числа:", числа) print(" ") брой
    = брой + 1

```

Изход:

Избор Сортиране на
 числа: [876, 356, 700, 247, 111] брой= 1 minIdx= 0 j0=
 1 числа: [111, 356, 700, 247, 876]

брой= 2 minIdx= 1 j0= 2 числа: [111,
 247, 700, 356, 876]

брой= 3 minIdx= 2 j0= 3 числа: [111,
 247, 356, 700, 876]

брой= 4 minIdx= 3 j0= 4 числа: [111,
 247, 356, 700, 876]

Край

Модул-24

Обединяване на списъци

Алгоритъмът за сливане приема два сортирани списъка като вход и произвежда един списък като изход, съдържащ всички елементи на двата входни списъка в сортиран ред.

Нека покажем процеса с помощта на реални данни:

Входни списъци:

```
list1 = [1, 3, 4, 5] list2 = [2, 4, 6, 8]
```

Изходен списък (обединен):

```
Списък 3 = [1, 2, 3, 4, 4, 5, 6, 8]
```

Примерът за гнездо показва реализация на два списъка. Ето кратък преглед:

При дадена итерация на първия цикъл, ние вземаме кой списък има елемент с по-малка стойност в техния индекс и напредваме този индекс. Този елемент ще заеме следващата позиция в обединения списък.

Накрая, след като прехвърлим всички елементи от един списък, ще копираме останалите от другия в обединения списък.

Пример 1

```
# Обединяване на два
списъка numbersA = [100, 103, 141, 150, 500, 900]
numbersB = [101, 140, 180, 910, 999] numbersX = []

print("numbersA:", numbersA)
print("numbersB:", numbersB)

# Стъпка 1.
idxA = 0
idxB = 0,
докато idxA < len(numbersA) и idxB < len(numbersB): # Проверете дали
текущият елемент от първия # списък е по-малък от текущия
елемент # от втория списък. Ако да, съхранете първия #
елемент от списък и увеличете първия списък # индекс. В
противен случай направете същото с втория списък, ако
numbersA[idxA] < numbersB[idxB]:

    # добавяне на глава(numbersA) към numbersX
    numbersX.append(numbersA[idxA]) idxA = idxA
    + 1 else: # добавяне на глава(numbersB) към
    numbersX numbersX.append(numbersB[idxB]) idxB =
    idxB + 1

print("Обединен списък(след стъпка 1):", numbersX)

# Стъпка 2.
# Досега само един от списъците е бил напълно копиран в numbersX # остава да
копирате другия входен списък

докато idxA < len(numbersA): #
останалите числаA трябва да бъдат копирани
numbersX.append(numbersA[idxA]) idxA = idxA + 1

докато idxB < len(numbersB): #
останалите числаB трябва да бъдат копирани
numbersX.append(numbersB[idxB]) idxB = idxB + 1

print("Обединен списък(окончателен):", numbersX)
```

Изход:

числаA: [100, 103, 141, 150, 500, 900] числаB: [101,
140, 180, 910, 999]
Обединен списък (след стъпка 1): [100, 101, 103, 140, 141, 150, 180, 500, 900]
Обединен списък (краен): [100, 101, 103, 140, 141, 150, 180, 500, 900, 910, 999]

Разгледайте сортирането на алгоритъма за сливане, реализиран като функция в скрипта:

Скрипт от ниво 3 на Python "Обединяване на два списъка"

Това е краят на НИВО-3; Следващите две дейности са следните:

а) Разгледайте примерите на Python ниво 3 (група А и В)

Връзка: www.windmaran.com/wm/python/py-main.php?px=L3A1

б) Отговорете на въпроси от теста за онлайн ниво 3 на

Python Връзка: www.windmaran.com/wm/python/py-quiz.php?px=L3Q1

След като завършите изучаването на това ръководство и попълните въпросите за онлайн викторина, вие сте готови да вземете тест за самооценка на Python. Има набор от задачи за тестване на вашите способности за кодиране:

Връзка: www.windmaran.com/wm/python/py-skill-test.php

Приложение 1

Стартирайте програми на Python с помощта на Thonny

Thonny е интегрирана среда за разработка (IDE) за Python, която е предназначена за начинаещи. Разгледайте и научете повече за Thonny на:

<https://thonny.org/>

Вижте онлайн урок на Thonny на:

<https://realpython.com/python-thonny/>

Как да изтеглите и инсталирате Thonny на вашия компютър, проверете началната страница на Thonny thonny.org

Тестване на Thonny Environment

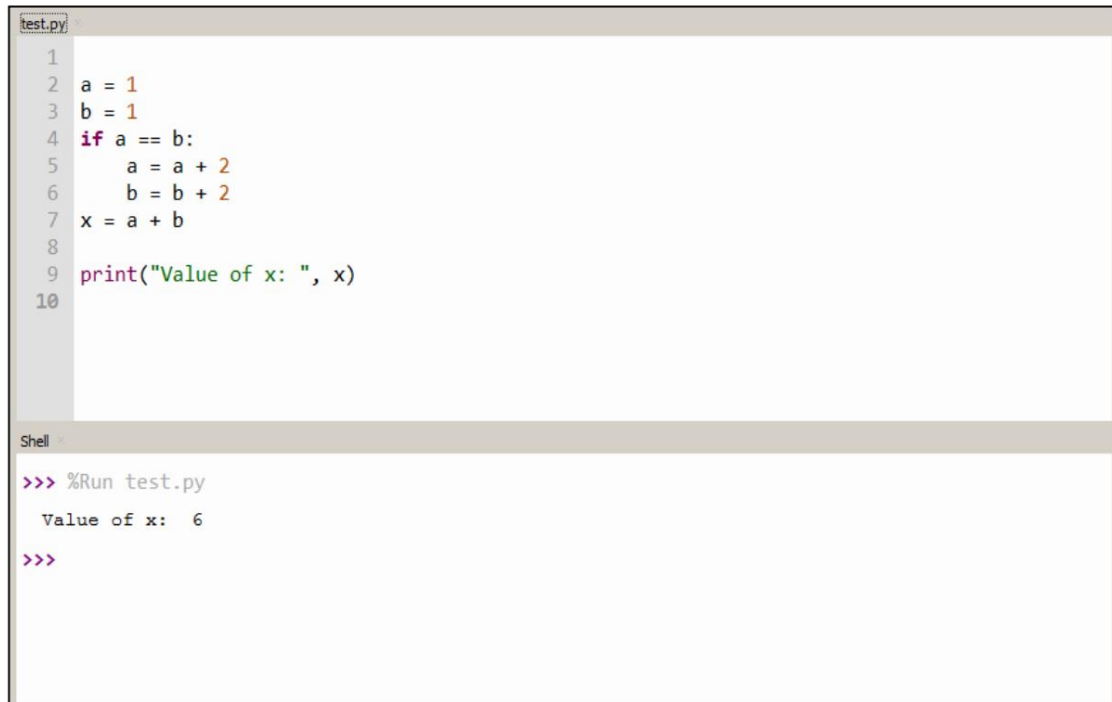
Ако приемем, че успешно сте изтеглили и инсталирали софтуера Thonny на вашия компютър, е време да стартирате първата програма. Ето една много проста програма, която може да се използва, за да се провери дали можем да изпълняваме програми на Python на нашия компютър:

Тестов код

```
a = 1 b
= 1,
ако a == b:
    a = a + 2 b =
    b + 2 x = a +
b

print("Стойност на x: ", x)
```

Когато копирате/поставите и изпълните горната програма, вашият екран на Thonny трябва да изглежда по следния начин:



The screenshot displays the Thonny Python IDE interface. The top pane, titled 'test.py', contains a Python script with the following code:

```
1
2 a = 1
3 b = 1
4 if a == b:
5     a = a + 2
6     b = b + 2
7 x = a + b
8
9 print("Value of x: ", x)
10
```

The bottom pane, titled 'Shell', shows the execution output:

```
>>> %Run test.py
Value of x:  6
>>>
```

Приложение-2

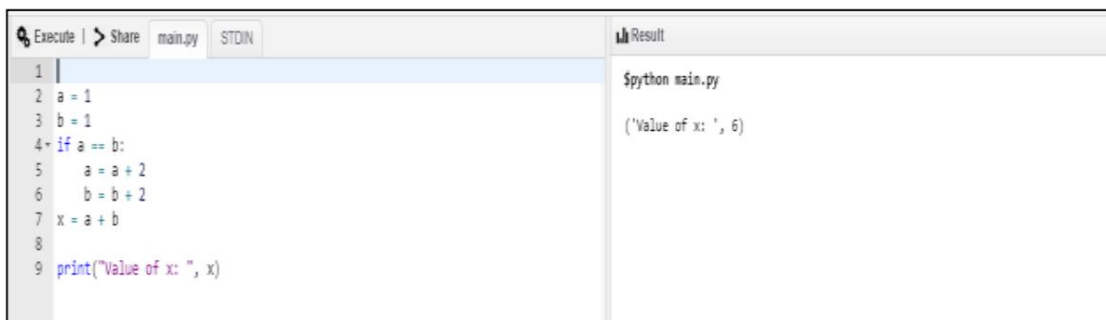
Стартирайте програми на Python онлайн в браузър

Има доста начини да стартирате Python във вашия уеб браузър. Препоръчваме тези уеб сайтове да разгледате

1. tutorialspoint.com

Отворете тази връзка: https://www.tutorialspoint.com/execute_python_online.php

С помощта на тестовия скрипт на Python, представен в Приложение-1, изпълнената програма изглежда така:



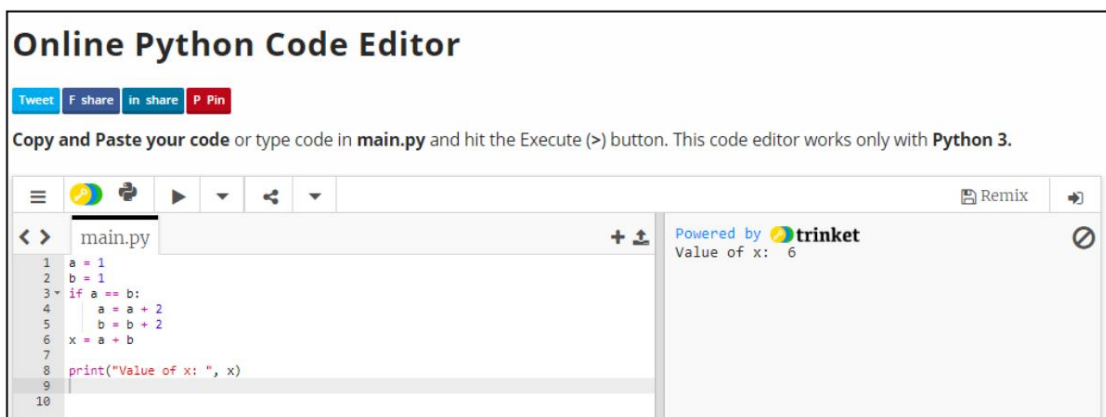
```
1
2 a = 1
3 b = 1
4 if a == b:
5     a = a + 2
6     b = b + 2
7 x = a + b
8
9 print('Value of x: ', x)
```

\$python main.py
('Value of x: ', 6)

2. pynative.com

Отворете онлайн редактор на програми и стартирайте тестовия код на python:

<https://pynative.com/online-python-code-editor-to-execute-python-code/>



Online Python Code Editor

Tweet F share in share P Pin

Copy and Paste your code or type code in **main.py** and hit the Execute (>) button. This code editor works only with **Python 3**.

```
1 a = 1
2 b = 1
3 if a == b:
4     a = a + 2
5     b = b + 2
6 x = a + b
7
8 print('Value of x: ', x)
9
10
```

Powered by trinket
Value of x: 6

Край