Технически Университет – София,
Факултет „Компютърни системи и технологии“

Упражнение 5 – Разработка на програми с използване на функции. Предаване на параметри. Връщане на резултат. Стойност по подразбиране.

Функции в Python

В програмирането функцията е самостоятелен блок код, който съдържа конкретна задача или група от свързани задачи. Ето някои вградени функции, предоставени от Python:

- **id ()- взема един аргумент и връща уникалния идентификатор на този обект;**
- **s = 'foobar'**
  print(id(s))
  139961200541936
- **len ()- връща дължината на аргумента;**
  a = ['foo', 'bar', 'baz', 'qux']
  print(len(a)) # 4
- any ()- приема аргумента като итерация и връща „True“, ако някой от елементите е „истина“ и „False“, ако не е;
  print(any([False, False, False])) # False
  print(any([False, True, False])) # True
  Пример 1
  print(any(['bar' == 'baz', len('foo') == 4, 'qux' in {'foo', 'bar', 'baz'}]))
  # False

  Пример 2
  print(any(['bar' == 'baz', len('foo') == 3, 'qux' in {'foo', 'bar', 'baz'}]))
  # True

Всяка от тези вградени функции изпълнява определена задача. Кодът, който изпълнява задачата, е дефиниран някъде, но не е нужно да се знае къде или дори как работи. Всичко, което трябва да се знае, е интерфейсът на функцията:

- Какви аргументи (ако има такива) са необходими
- Какви стойности (ако има такива) връща

След това се извиква функцията и се подават съответните аргументи. Изпълнението на програмата отива в определената част от кода. Когато функцията приключи, изпълнението се връща към кода, от мястото, където е спрял. Функцията може да връща или да не връща данни, които кодът да използва.

Когато се дефинира собствена функция на Python, тя работи по същия начин. От някъде в кода ще бъде извикана съответната функция на Python и изпълнението на програмата ще се прехвърли в тялото на кода, който съставя функцията. Когато функцията приключи, изпълнението се връща на мястото, където функцията е била извикана. В зависимост от това как е проектиран интерфейса на функцията, данните могат да се предават при извикване на функцията, а връщаните стойности могат да се предават обратно, когато приключи. Обичайният синтаксис за дефиниране на функция в Python е следният:

def <function_name>([<parameters>]):

<statement(s)>

def- ключовата дума, която информира Python, че се дефинира функция;

<function_name>- идентификатор на Python, който именува функцията;

<parameters>- (по избор) списък с параметри, разделени със запетая, които могат да бъдат подаени към функцията;

:- пунктуация, която обозначава края на хедъра (header) на функцията (списък с имена и параметри)

<statement (s)>- блок от валидни твърдения (statements) на Python

Последният елемент, <statement (s)>, се нарича тяло (body) на функцията. Тялото е блок от твърдения (statements), които ще бъдат изпълнени при извикване на функцията.

Синтаксисът за извикване на функция на Python е следният:

<function_name>([<arguments>])

<arguments>- стойностите, подадени към функцията. Те съответстват на <parameters> в дефинирането на функцията. Може да се дефинирате функция,

която не приема никакви аргументи, но скобите са задължителни. И дефинирането на функция, и извикването на функция винаги трябва да включва скоби, дори и да са празни.

Пример:

```
def f():
    s = '-- Inside f()'
    print(s)
print('Before calling f()')
f()
print('After calling f()')
```

Предаване на аргументи

Позиционни аргументи

Най -простият начин за предаване на аргументи в една функция на Python е с позиционни аргументи (наричани още задължителни аргументи). В дефинирането на функцията се посочва разделен със запетая списък на параметрите в скобите:

```
def f(qty, item, price):
    print(f'{qty} {item} cost ${price:.2f}')
```

Когато функцията е извикана, се посочва съответния списък с аргументи:

```
f(6, 'bananas', 1.74)  # 6 bananas cost $1.74
```

Параметрите (qty, item, price) се държат като променливи, които са дефинирани локално за функцията. Когато функцията е извикана, аргументите, които се предават (6, 'bananas', 1.74) отговарят на параметрите по ред, подобно на присвояване на променлива. Въпреки че позиционните аргументи са най простият начин за предаване на данни към функция, те също така позволяват най -малко гъвкавост. Като начало редът на аргументите в извикването трябва да съвпада с реда на параметрите в дефинирането. При позиционни аргументи, аргументите в извикването и параметрите в дефинирането трябва да се съгласуват не само по ред, но и по брой. Това е причината позиционните аргументи да се наричат и задължителни. Не можете да пропуснете нищо при извикване на функцията:

```
# Too few arguments
f(6, 'bananas')
Traceback (most recent call last):
```

```
  File "main.py", line 3, in <module>
    f(6, 'bananas')
TypeError: f() missing 1 required positional argument: 'price'
```

f(6, 'bananas') — това е извикване на функция f с два аргумента: 6 и 'bananas'.

Traceback (most recent call last): ... — това е стандартен Python отчет за грешка. Показва къде в изпълнението е възникнала проблема (тук — в интерактивния shell, ред 1).

TypeError: f() missing 1 required positional argument: 'price' — сама по себе си това е важната част:

- **TypeError** е вида на грешката (тип грешка).
- Съобщението казва, че функцията f е дефинирана така, че **изисква** един допълнителен (трети) позиционен аргумент с име price, но при извикването ти е подала само два. Python чака трета стойност и когато я няма хвърля тази грешка.

Нито може да се посочат допълнителни:

```
# Too many arguments
f(6, 'bananas', 1.74, 'pears')
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    f(6, 'bananas', 1.74, 'pears')
TypeError: f() takes 3 positional arguments but 4 were given
```

Това съобщение означава, че е извикана функция f, която е дефинирана да приема **3 позиционни аргумента**, а ние сме ѝ подали **4**, затова Python вдига TypeError.

```
f(6, 'bananas', 1.74)   # правилно
```

Аргументи- ключови думи

Когато се извиква функция, може да се посочат аргументи във формата <keyword> = <value>. В този случай всяка <ключова дума(keyword)> трябва да съответства на параметър в дефинирането на функцията на Python. Например, предварително дефинираната функция f () може да бъде извикана с ключови аргументи, както следва:

```
f(qty=6, item='bananas', price=1.74)
# -> 6 bananas cost $1.74
```

Използването на ключова дума, която не съответства на нито един от декларираните параметри, генерира изключение:

```
f(qty=6, item='bananas', cost=1.74)

Traceback (most recent call last):

  File "main.py", line 4, in <module>

    f(qty=6, item='bananas', cost=1.74)

TypeError: f() got an unexpected keyword argument 'cost'
```

Параметри по подразбиране

Ако параметър, посочен в дефинирането на функция на Python, има формата <name>=<value>, тогава <value> става стойност по подразбиране за този параметър. Параметрите, дефинирани по този начин, се наричат параметри по подразбиране или опционални:

```
def f(qty=6, item='bananas', price=1.74):

    print(f'{qty} {item} cost ${price:.2f}')

f(4, 'apples', 2.24)     # 4 apples cost $2.24

f(4, 'apples')           # 4 apples cost $1.74

f(4)                     # 4 bananas cost $1.74

f()                      # 6 bananas cost $1.74

f(item='pears', qty=9)  # 9 pears cost $1.74

f(price=2.29)            # 6 bananas cost $2.29
```

Позиционните агументи трябва да отговарят по ред и номер с параметрите, декларирани при дефинирането на функцията.

Аргументите- ключови думи трябва да съвпадат с декларираните параметри по номер, но могат да бъдат посочени в произволен ред.

Параметрите по подразбиране позволяват някои аргументи да бъдат пропуснати при извикване на функцията.

Връщане на резултат

Една от възможностите е да се използва function return value. Return statement в една функция на Python има две приложения:

- незабавно прекратява функцията и предава управлението на изпълнението обратно на викащия функцията;
- осигурява механизъм, чрез който функцията може да предава данни обратно на викащия функцията.

Излизане от функция

„Return“ предизвиква незабавно излизане от функцията и предава управлението на изпълнението обратно на викащия функцията:

```
def f():
   print('foo')
   print('bar')
   return  # връща None, може да се изпусне
f()      # foo
           bar
```

В този пример „return“ е излишен. Функция ще се върне към викащия, когато стигне до края - тоест след изпълнението на последното твърдение на тялото на функцията. Така че тази функция би се държала идентично и без „return“. “Return statement” обаче не трябва да бъде в края на функцията. Може да се появи навсякъде в тялото на функцията и дори няколко пъти:

```
def f(x):
   if x < 0:
      return
   if x > 100:
```

```
        return
    print(x)
f(-3)
f(105)
f(64)           # 64
```

Първите две извиквания на функцията f () не дават резултат на изхода, тъй като „return“ се изпълнява и функцията излиза преждевременно, преди да се достигне „print ()“ на ред 6. Това може да бъде полезно за проверка на грешки във функцията. Могат да се проверят няколко условия за грешка в началото на функцията, като „return“ ще сигнализира, ако има проблем:

```
 def f(error_cond1=False, error_cond2=False, error_cond3=False):
    if error_cond1:
        return
    if error_cond2:
        return
    if error_cond3:
        return
    # normal processing
    print("Нормална обработка...")

# извикване
f()                     # -> Нормална обработка...
f(error_cond2=True)     # -> (няма отпечатване)
```

Ако не са налице нито едно от условията за грешка, тогава функцията може да продължи с нормалната си обработка.

Връщане на данни към викащия функцията

„Return“ също се използва и за предаване на данни обратно към викащия. Ако „return“ във функция на Python е последвана от израз, тогава в извикващата среда извикването на функцията оценява стойността на този израз:

```
def f():
    return 'foo'
s = f()
print(s)   #  foo
```

Тук стойността на израза f ()е „foo“, която впоследствие се присвоява на променлива s.

Функцията може да върне всеки тип обект. В Python това означава почти всичко. В извикващата среда извикването на функция може да се използва синтактично по всякакъв начин, който има смисъл за типа обект, който функцията връща.

Например в този код f () връща речник. В извикващата среда тогава изразът f () представлява речник, а f () ['baz'] е валидна ключова референция в този речник:

```
def f():
    return dict(foo=1, bar=2, baz=3)

# извикване
print(f())          # -> {'foo': 1, 'bar': 2, 'baz': 3}

# достъп до отделен елемент
print(f()['baz'])   # 3
```

Функции с променлив брой аргументи

в Python има възможност да бъде създадена функция с променлив брой параметри. За целта използваме аргумента *

Пример :

```
def psum(*k):
   result=0
   for i in k:
      result+=i
   return result
s=psum(1,2,3,4)
print(s)       #10
```

def psum(*k):

- *k означава: „вземи всички предадени позиционни аргументи и ги сложи в един пакет“ — в Python този пакет е **кортеж**. Името k е произволно и се записва с *k.

result = 0

- Инициализираме променлива, в която ще трупаме сумата.

for i in k:

- Обхождаме всеки елемент от кортежа k. Ако сме извикали psum(1,2,3,4), то k == (1, 2, 3, 4) и цикълът ще премине през 1, после 2, после 3, после 4.

result += i

- Това е кратка форма за result = result + i — добавяме текущия елемент към сумата.

return result

- След като обходим всички елементи, връщаме натрупаната сума.

s = psum(1,2,3,4) → функцията връща 10, print(s) отпечатва 10.

Анонимни функции -lambda

Може да бъде създадена функция без име , чиято цел е да изчисли стойността на някакъв израз. Описанието на такава функция започва с ключовата дума lambda , след което се изброяват аргументите, поставя се двоеточие и се указва стойността връщана от функцията. Стойността на такава инструкция се явява референция към обект на функция. Такава референция може да бъде присвоена в качеството на стойност на променлива.

lambda аргументи:резултат

Ако аргументите са няколко те се изброяват със запетая. Ако няма аргументи те просто се пропускат.

Пример : **num=10**

```
#функция на основата на ламбда израз
L=lambda n:2*n+1
#проверка на резултат
print('нечетни числа:')
for k in range(num):
    print(L(k),end=' ')
#директно извикване на ламбда функция
print('\n квадрати на числата:')

for k in range(num):
    print((lambda x:x*x)(k+1),end=' ')
```

резултат от изпълнението:

```
Нечетни числа:
1 3 5 7 9 11 13 15 17 19
Квадрати на числата:
1  4  9  16  25  36  49  64  81  100
```

Локални и глобални променливи

Променливите които създаваме във функциите се наричат локални и са достъпни само в тялото на функцията. Променливи , които се създават извън тялото на функциите се наричат глобални променливи. Важно е правилото , че ако в тялото на функция се присвоява стойност на променлива, то такава променлива се интерпретира като локална дори ако съществува глобална променлива със същото име. Ако стойността на променливата само се прочита , тогава се използва глобалната променлива. Ако в тялото на функцията искаме да използваме глобална променлива (в това число да и присвояваме стойност) трябва да я декларираме с ключовата дума **global**.

Пример :

```
def func():
   global var1
var1=10
print(var1)  #10
```

Задачи:

1. Напишете програма, която намира лицето на геометрична фигура като първо се въвежда вида на фигурата:

1- квадрат
2-правоъгълник
3- прав.триъгълник

За пресмятане на лицето на отделните фигури да се напишат подходящи функции.

2. Напишете потребителска функция , проверяваща дали число е палиндром. Функцията получава като аргумент число, връща като резултат 1, ако числото е палиндром и 0 ако числото не е палиндром.

3. Програма, която реализира калкулатор за цели числа. Действията които изпълнява са : Събиране +
Изваждане –
Умножение *
Деление /
Потребителя въвежда вида на операцията.

После въвежда две числа и резултата се извежда на екрана . Реализирайте отделни функции за отделните операции.

4. На функция се подават два аргумента: списък с числа и число. Променете всички елементи от списъка със стойност по-голяма от даденото число на 0(нула).