Python programming for beginners

Stefan Zhauryd
Instructor

Module 4

Functions, Tuples, Dictionaries, and Data Processing

In this module, you will learn about:

- code structuring and the concept of function;
- **function invocation and returning a result from a function;**
- **name scopes and variable shadowing;**
- tuples and their purpose, constructing and using tuples;
- dictionaries and their purpose, constructing and using dictionaries.

Effects and results: the return instruction

```
return
```

```
a = max(1,2,3,4,5)

def max(....):
    return 5

s = input()
```

return without an expression

```
def happy_new_year(wishes = True):
    print("Three...")
    print("Two...")
    print("One...")
    if not wishes:
        return

    print("Happy New Year!")
```

```
happy_new_year()
```

```
Three...
Two...
One...
Happy New Year!
```

```
happy_new_year(False)
```

```
Three...
Two...
One...
```

```
return
```

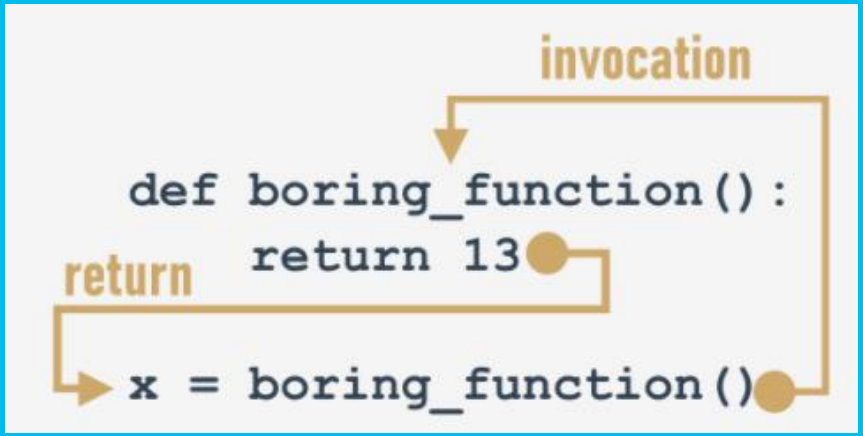
return with an expression

```
def boring_function():
    return 123

x = boring_function()

print("The boring_function has returned its result. It's:", x)
```

```
The boring_function has returned its result. It's: 123
```




```
def boring_function():
    print("'Boredom Mode' ON.")
    return 123

print("This lesson is interesting!")
boring_function()
print("This lesson is boring...")
```

```
This lesson is interesting!
'Boredom Mode' ON.
This lesson is boring...
```

A few words about None

```
print(None + 2)

```

```
Traceback (most recent call last):
  File "main.py", line 1, in <module>
    print(None + 2)
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

Its data doesn't represent any reasonable value - actually, it's not a value at all; hence, it mustn't take part in any expressions.

Note: **None is a keyword.**

There are only two kinds of circumstances when None can be safely used:

- **when you assign it to a variable** (or return it as a function's result)
- **when you compare it with a variable to diagnose its internal state.**

```
value = None
if value is None:
    print("Sorry, you don't carry any value")

```

```
Console >_
Sorry, you don't carry any value
```

A few words about None: continued

```
def strange_function(n):
    if(n % 2 == 0):
        return True

print(strange_function(2))
print(strange_function(1))
print()

print(strange_function(8))
print(strange_function(5))
print()

print(strange_function(10))
print(strange_function(53))
print()
```

Console >_

```
True
None

True
None

True
None
```

Effects and results: lists and functions

```
def list_sum(lst):
    s = 0

    for elem in lst:
        s += elem

    return s


print(list_sum([5, 4, 3]))
print()

print(list_sum(5))
```

Console >_

```
12

Traceback (most recent call last):
  File "main.py", line 13, in <module>
    print(list_sum(5))
  File "main.py", line 4, in list_sum
    for elem in lst:
TypeError: 'int' object is not iterable
```

Effects and results: lists and functions - continued

```python
def strange_list_fun(n):
    strange_list = []

    for i in range(0, n):
        strange_list.insert(0, i)

    return strange_list

print(strange_list_fun(5))
```

Console >_

```
[4, 3, 2, 1, 0]
```

Home work 5.1
A leap year

Your task is to write and test a function which **takes one argument (a year)** and **returns True if the year is a leap year**, or **False otherwise.**

The seed of the function is already sown in the skeleton code in the next slide.

Note: I've also prepared a short testing code, which you can use to test your function.

The code uses two lists - one with the test data, and the other containing the expected results. The code will tell you if any of your results are invalid.

Home work 5.1
A leap year

```python
def is_year_leap(year):
#
# put your code here
#

test_data = [1900, 2000, 2016, 1987]
test_results = [False, True, True, False]
for i in range(len(test_data)):
    yr = test_data[i]
    print(yr,"->",end="")
    result = is_year_leap(yr)
    if result == test_results[i]:
        print("OK")
    else:
        print("Failed")

```

Home work 5.2 Converting

A car's fuel consumption may be expressed in many different ways. For example, in Europe, it is shown as the amount of fuel consumed per 100 kilometers.
In the USA, it is shown as the number of miles traveled by a car using one gallon of fuel.
Your task is to write a pair of functions converting liters/100km into miles/gallons, and vice versa.

The functions:

- are named **liters_100km_to_miles_gallon** and **miles_gallon_to_liters_100km** respectively;
- take one argument (the value corresponding to their names)

Here is some information to help you:
1 American mile = 1609.344 metres;
1 American gallon = 3.785411784 litres.

Home work 5.2 Converting

American_mile = 1609.344
American_gallon = 3.785411784

gall = liters / amer_gall
miles = 100 * 1000 / amer_mil
return miles/gall

```python
def liters_100km_to_miles_gallon(liters):
#
# Write your code here.
#

def miles_gallon_to_liters_100km(miles):
#
# Write your code here
#

print(liters_100km_to_miles_gallon(3.9))
print(liters_100km_to_miles_gallon(7.5))
print(liters_100km_to_miles_gallon(10.))
print(miles_gallon_to_liters_100km(60.3))
print(miles_gallon_to_liters_100km(31.4))
print(miles_gallon_to_liters_100km(23.5))
```

Console >_

```
60.31143162393162
31.36194444444444
23.52145833333333
3.9007393587617467
7.490910297239916
10.009131205673757
```

```
def hi():
    return
    print("Hi!")

hi()
```

Examples

```
def is_int(data):
    if type(data) == int:
        return True
    elif type(data) == float:
        return False

print(is_int(5))
print(is_int(5.0))
print(is_int("5"))
```

```
def even_num_lst(ran):
    lst = []
    for num in range(ran):
        if num % 2 == 0:
            lst.append(num)
    return lst

print(even_num_lst(11))
```

```
def list_updater(lst):
    upd_list = []
    for elem in lst:
        elem **= 2
        upd_list.append(elem)
    return upd_list

foo = [1, 2, 3, 4, 5]
print(list_updater(foo))
```

Examples

```python
# Example 1
def wishes():
    print("My Wishes")
    return "Happy Birthday"

wishes()     # outputs: My Wishes


# Example 2
def wishes():
    print("My Wishes")
    return "Happy Birthday"

print(wishes())

# outputs: My Wishes
#          Happy Birthday
```

```python
def hil(my_list):
    my_list[1] = 1111
    print("Inside the func:", my_list)

a = ["Adam", "John", "Lucy", "Goose"]
print("Original list:", a)
hil(a)
print("Orig after proc:", a)
print()

a = ["Adam", "John", "Lucy", "Goose"]
print("Original list:", a)
hil(a[:])
print("Orig after proc:", a)
print()

a = ["Adam", "John", "Lucy", "Goose"]
print("Original list:", a)
hil(a[:2])
print("Orig after proc:", a)
print()
```

```
====== RESTART: D:/Python content/002 week/week 2 I
Original list: ['Adam', 'John', 'Lucy', 'Goose']
Inside the func: ['Adam', 1111, 'Lucy', 'Goose']
Orig after proc: ['Adam', 1111, 'Lucy', 'Goose']

Original list: ['Adam', 'John', 'Lucy', 'Goose']
Inside the func: ['Adam', 1111, 'Lucy', 'Goose']
Orig after proc: ['Adam', 'John', 'Lucy', 'Goose']

Original list: ['Adam', 'John', 'Lucy', 'Goose']
Inside the func: ['Adam', 1111]
Orig after proc: ['Adam', 'John', 'Lucy', 'Goose']

>>>
```

Functions and scopes

```python
def scope_test():
    x = 123


for i in range(3):
    print(i)

print(i)



scope_test()
print(x)  #NameError: name 'x' is not defined
```

Console >_

```
0
1
2
2
Traceback (most recent call last):
  File "main.py", line 13, in <module>
    print(x)  #NameError: name 'x' is not defined
NameError: name 'x' is not defined
```

Functions and scopes: continued **id(var)**

```python
def my_function():
    var = 222
    print("Do I know that variable?", var)


var = 1
my_function()
print(var)
```

Console >_

```
Do I know that variable? 222
1
```

```python
def f():
    print("Do I know that var? ", var)


var = 100
print(var)

print(f())
```

global

Functions and scopes: the global keyword

global nonlocal

```
global name
global name1, name2, ...
```

```
def my_function():
    global var
    var = 2
    print("Do I know that variable?", var)


var = 1
my_function()
print(var)
```

Console >_

```
Do I know that variable? 2
2
```

```python
def my_function(n):
    print("I got", n)
    n += 1
    print("I have", n)


var = 1
my_function(var)
print(var)
```

Console >_

```
I got 1
I have 2
1
```

```python
def my_function(my_list_1):
    print("Print #1:", my_list_1)
    print("Print #2:", my_list_2)
    my_list_1 = [0, 1]
    print("Print #3:", my_list_1)
    print("Print #4:", my_list_2)


my_list_2 = [2, 3]
my_function(my_list_2)
print("Print #5:", my_list_2)
```

Console >_

```
Print #1: [2, 3]
Print #2: [2, 3]
Print #3: [0, 1]
Print #4: [2, 3]
Print #5: [2, 3]
```

How the function interacts with its arguments

```python
def my_function(my_list_1):
    print("Print #1:", my_list_1)
    print("Print #2:", my_list_2)
    del my_list_1[0]  # Pay attention to this line.
    print("Print #3:", my_list_1)
    print("Print #4:", my_list_2)


my_list_2 = [2, 3]
my_function(my_list_2)
print("Print #5:", my_list_2)
```

Console >_

```
Print #1: [2, 3]
Print #2: [2, 3]
Print #3: [3]
Print #4: [3]
Print #5: [3]
```

Key t

```python
var = 2


def mult_by_var(x):
    return x * var


print(mult_by_var(7))    # outputs: 14
```

```python
def mult(x):
    var = 7
    return x * var


var = 3
print(mult(7))    # outputs: 49
```

```python
def mult(x):
    var = 5
    return x * var


print(mult(7))    # outputs: 35
```

```python
def adding(x):
    var = 7
    return x + var


print(adding(4))    # outputs: 11
print(var)    # NameError
```

```python
var = 2
print(var)    # outputs: 2


def return_var():
    global var
    var = 5
    return var


print(return_var())    # outputs: 5
print(var)    # outputs: 5
```

Examples

```
a = 1


def fun():
    a = 2
    print(a)


fun()
print(a)
```

```
def message():
    alt = 1
    print("Hello, World!")


print(alt)
```

Examples

```
a = 1


def fun():
    global a
    a = 2
    print(a)


fun()
a = 3
print(a)
```

```
a = 1


def fun():
    global a
    a = 2
    print(a)


a = 3
fun()
print(a)
```

```
def fun():
    global var
    var = 100
    print(var, 'inside')

fun()
print(var, 'outside')

var = 20
print(var, 'new')
```

Some simple functions: evaluating the BMI

As you can see, the formula gets two values:

- **weight (originally in kilograms)**
- **height (originally in meters)**

```python
def bmi(weight, height):
    if weight or height <= 0:
        return None

    return weight / height ** 2


print(bmi(0, 0))
print(bmi(4, 4))
print(bmi(-99, 1.65))
print(bmi(0, -10))
print(bmi(52.5, 1.65))
```

Console >_

```
None
None
None
None
```

As you can see, the formula gets two values:

- **weight (originally in kilograms)**
- **height (originally in meters)**

Some simple functions: evaluating the BMI

```python
def bmi(weight, height):
    return weight / height ** 2


print(bmi(52.5, 1.65))
```

Console >_

```
19.283746556473833
```

Some simple functions: evaluating BMI and converting imperial units to metric units

```
def lb_to_kg(lb):
    return lb * 0.45359237


print(lb_to_kg(1))
```

```
0.45359237
```

```
def ft_and_inch_to_m(ft, inch):
    return ft * 0.3048 + inch * 0.0254


print(ft_and_inch_to_m(1, 1))
```

```
0.3302
```

```
def ft_and_inch_to_m(ft, inch = 0.0):
    return ft * 0.3048 + inch * 0.0254


print(ft_and_inch_to_m(6))
```

```
1.8288000000000002
```

```
def ft_and_inch_to_m(ft, inch = 0.0):
    return ft * 0.3048 + inch * 0.0254


def lb_to_kg(lb):
    return lb * 0.45359237


def bmi(weight, height):
    if height < 1.0 or height > 2.5 or weight < 20 or weight > 200:
        return None

    return weight / height ** 2


print(bmi(weight = lb_to_kg(176), height = ft_and_inch_to_m(5, 7)))
```

```
27.565214082533313
```

```
def bmi(weight, height):
    if height < 1.0 or height > 2.5 or \
    weight < 20 or weight > 200:
        return None

    return weight / height **


print(bmi(352.5, 1.65))
```

Console >_

```
None
```

```python
def is_a_triangle(a, b, c):
    if a + b <= c:
        return False
    if b + c <= a:
        return False
    if c + a <= b:
        return False
    return True


print(is_a_triangle(1, 1, 1))
print(is_a_triangle(1, 1, 3))
```

Console >_

```
True
False
```

Some simple functions: continued

```python
def is_a_triangle(a, b, c):
    if a + b <= c or b + c <= a or c + a <= b:
        return False
    return True


print(is_a_triangle(1, 1, 1))
print(is_a_triangle(1, 1, 3))
```

```python
def is_a_triangle(a, b, c):
    return a + b > c and b + c > a and c + a > b


print(is_a_triangle(1, 1, 1))
print(is_a_triangle(1, 1, 3))
```

Some simple functions: triangles and the Pythagorean theorem

```python
def is_a_triangle(a, b, c):
    return a + b > c and b + c > a and c + a > b


a = float(input('Enter the first side\'s length: '))
b = float(input('Enter the second side\'s length: '))
c = float(input('Enter the third side\'s length: '))

if is_a_triangle(a, b, c):
    print('Yes, it can be a triangle.')
else:
    print('No, it can\'t be a triangle.')
```

```python
def is_a_right_triangle(a, b, c):
    if not is_a_triangle(a, b, c):
        return False
    if c > a and c > b:
        return c ** 2 == a ** 2 + b ** 2
    if a > b and a > c:
        return a ** 2 == b ** 2 + c ** 2
```

Console >_

```
Enter the first side's length: 10
Enter the second side's length: 5
Enter the third side's length: 8
Yes, it can be a triangle.
Enter the first side's length: 10
Enter the second side's length: -2
Enter the third side's length: 3
No, it can't be a triangle.
```

```python
def is_a_triangle(a, b, c):
    return a + b > c and b + c > a and c + a > b


a = float(input('Enter the first side\'s length: '))
b = float(input('Enter the second side\'s length: '))
c = float(input('Enter the third side\'s length: '))

if is_a_triangle(a, b, c):
    print('Yes, it can be a triangle.')
else:
    print('No, it can\'t be a triangle.')
```

Console >_

```
Enter the first side's length: 3
Enter the second side's length: 2
Enter the third side's length: 3
Yes, it can be a triangle.
```

Some simple functions: evaluating a triangle's area

$$s = \frac{a+b+c}{2} \qquad A = \sqrt{s(s-a)(s-b)(s-c)}$$

```python
def is_a_triangle(a, b, c):
    return a + b > c and b + c > a and c + a > b


def heron(a, b, c):
    p = (a + b + c) / 2
    return (p * (p - a) * (p - b) * (p - c)) ** 0.5


def area_of_triangle(a, b, c):
    if not is_a_triangle(a, b, c):
        return None
    return heron(a, b, c)


print(area_of_triangle(1., 1., 2. ** .5))
```

Console >_

```
0.49999999999999983
```

Home work 5.3 Factorial

implement program to calculate factorial **without recursion**

```
def factorial(n)
        if n < 0: ret None
        if n < 2: ret 1

factor_val = 1
for (2, n+1)
        factor_val*=i

return factor_val
```

```
0! = 1 (yes! it's true)
1! = 1
2! = 1 * 2
3! = 1 * 2 * 3
4! = 1 * 2 * 3 * 4
:
:
n! = 1 * 2 ** 3 * 4 * ... * n-1 * n
```

```
for i in range(-1, 9):
    print(factorial(i))
```

```
None
1
1
2
6
24
120
720
5040
40320
>>>
```

Home work 5.4 Fibonacci numbers

Def febo(n)

n < 1 --> None

n < 3 --> 1

f1,f2 = 1,1

for I in…..

f3 = f1 +f2 #2

f1, f2= f2, f3

--> f3

```
fib_1 = 1
fib_2 = 1
fib_3 = 1 + 1 = 2
fib_4 = 1 + 2 = 3
fib_5 = 2 + 3 = 5
fib_6 = 3 + 5 = 8
fib_7 = 5 + 8 = 13
```

```
None
None
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
10946
17711
28657
46368
```

- They are a sequence of integer numbers built using a very simple rule:
- the first element of the sequence is equal to one ($Fib_1 = 1$)
- the second is also equal to one ($Fib_2 = 1$)
- every subsequent number is the the_sum of the two preceding numbers: ($Fibi = Fibi_1 + Fibi_2$)

- Ask end number of sequences.***
- find each febo

Use:

- int() input() and so on***

```
for i in range(-1, 25):
    print(febo(i))
```

Some simple functions: recursion

```
def fib(n):
    if n < 1:
        return None
    if n < 3:
        return 1

    elem_1 = elem_2 = 1
    the_sum = 0
    for i in range(3, n + 1):
        the_sum = elem_1 + elem_2
        elem_1, elem_2 = elem_2, the_sum
    return the_sum


for n in range(1, 10):
    print(n, "->", fib(n))
```

Console >_

```
1 -> 1
2 -> 1
3 -> 2
4 -> 3
5 -> 5
6 -> 8
7 -> 13
8 -> 21
9 -> 34
```

```
def febo(n):
    if n < 1:
        return None
    if n < 3:
        return 1
    return febo(n - 1) + febo(n - 2)

for n in range(1, 10):
    print(n, "->", febo(n))
```

```
1 -> 1
2 -> 1
3 -> 2
4 -> 3
5 -> 5
6 -> 8
7 -> 13
8 -> 21
9 -> 34
>>>
```

https://bit.ly/3Cjr03G

Key takeaways

```
# Recursive implementation of the factorial function.

def factorial(n):
    if n == 1:    # The base case (termination condition.)
        return 1
    else:
        return n * factorial(n - 1)


print(factorial(4)) # 4 * 3 * 2 * 1 = 24
```


1. A function can call other functions or even itself. When a function calls itself, this situation is known as recursion, and the function which calls itself and contains a specified termination condition (i.e., the base case - a condition which doesn't tell the function to make any further calls to that function) is called a recursive function.

2. You can use recursive functions in Python to write clean, elegant code, and divide it into smaller, organized chunks. On the other hand, you need to be very careful as it might be easy to make a mistake and create a function which never terminates. You also need to remember that recursive calls consume a lot of memory, and therefore may sometimes be inefficient.

https://bit.ly/3Cjr03G

Example

```
def fun(a):
    if a > 30:
        return 3
    else:
        return a + fun(a + 3)


print(fun(25))
```

(fun(25) --> **25** + fun(28) --> **28** +fun(31) --> **3 = 56**

25 28 3

http://www.pythontutor.com/visualize.html#mode=display

ЗАДАНИЯ

1) Прорешать всю классную работу
2) Выполнить все домашние задания

Почитать:

1) Byte of Python – Прочитать страницы - стр. 64-75

Крайний срок сдачи 05/10 в 21:00 (можно раньше, но не позже)

ЗАДАНИЯ

Название файлов, которые вы отправляете мне в telegram:

Vasia_Pupkin_class_work_L5_P1.py

+все задания ОДНИМ ФАЙЛОМ - Vasia_Pupkin_L5_P1.py

Формат сообщения которое вы присылаете мне

(после полного выполнения домашнего задания, только один раз) в Telegram:

Добрый день/вечер. Я Вася Пупкин, и это мои домашние задания к лекции 5 часть 1 про функции.

И отправляете файлы

Крайний срок сдачи 05/10 в 21:00 (можно раньше, но не позже)

https://docs.github.com/articles/using-pull-requests

Create your possibilities. Bye bye.