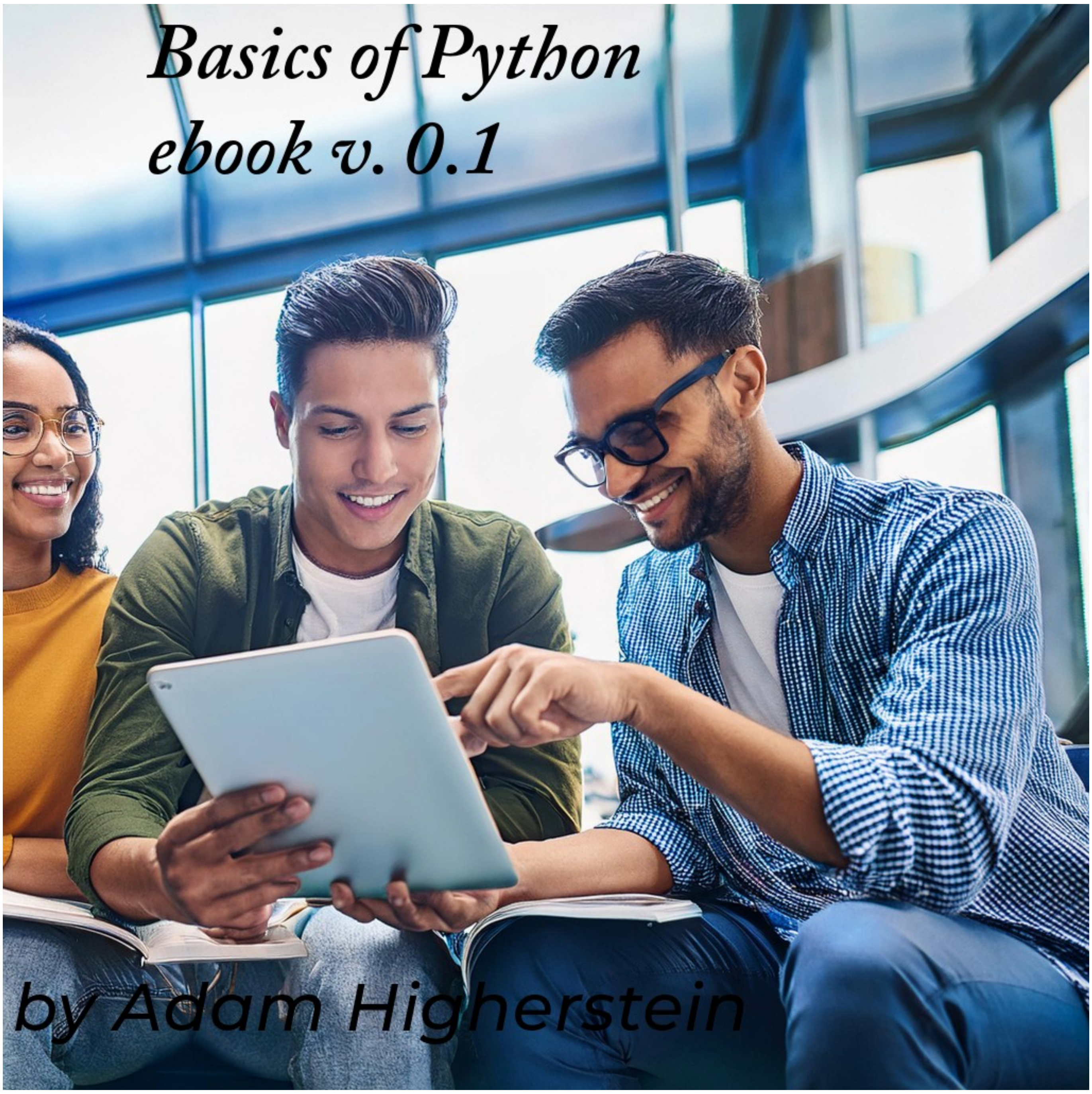
Basics of Python ebook v. 0.1

by Adam Higherstein



Basics in programming with Python

Free eBook

by Adam

Table of Contents

Developing Python Apps

Introduction

What is programming?

We give instructions to the computer: set of instructions is a program.
Computer is mainly the processor that can understand machine code.
So our instructions are compiled to machine code so that it can be executed by the computer.
A program contains
storages, data structures
functions, activities, operations

Programming languages
There are several programming languages, also for different purposes.
Here are the most used languages:
Java
Used in workstation and enterprise applications AND Android phones
It is also an Object Oriented Programming language (OOP)

C and C++
C is used in Embedded programming, games and so on
Procedural Programming language (not OOP!!)
C++
It is also an Object Oriented Programming language (OOP)
C++ is used in Game programming for different kinds of applications and for operating systems

C#
We use C# in this book. It is developed by Microsoft, used in ASP.NET, workstation software, games etc.
It is also an Object Oriented Programming language (OOP)

PHP
Used for web programming
JavaScript
Used for web programming

HTML
Web page contents markup language

Python
Used for different kinds of applications

SQL
Used for database queries

ObjectiveC
Used for IOS plattform

Assembler
It is a symbolic machine language

Take a look at the lists of programming languages. Here is for example a list of most popular programming languages:
http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages

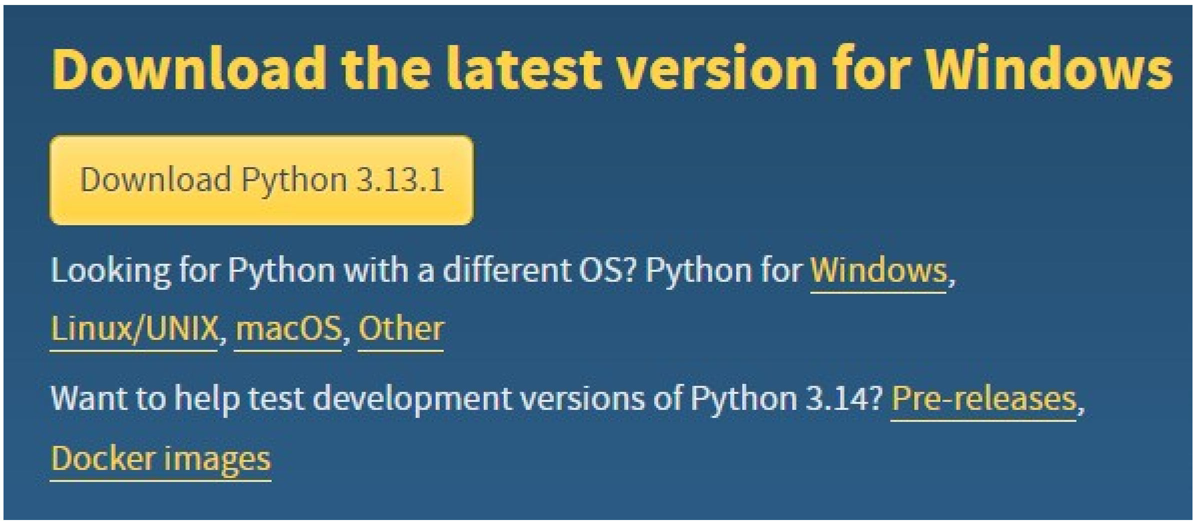
Next we take a look at our tool!
When you install Python to your machine, you get also Idle-tool.
There are several other tool, also: for example PyCharm is very popular.
Also, Jupyter Notebook is used a lot.

Install Python

Newest Python version is Python 3.13 (december, 2024).

Because we concentrate on basics, you can use also previous Python versions!
You can download Python e.g. from this place:

https://www.python.org/downloads/



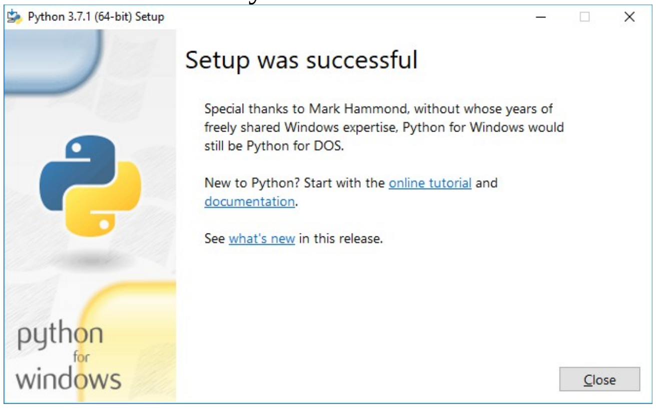
Download the latest version for Windows

Download Python 3.13.1

Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other

Want to help test development versions of Python 3.14? Pre-releases, Docker images

AND then install Python:



Python 3.7.1 (64-bit) Setup

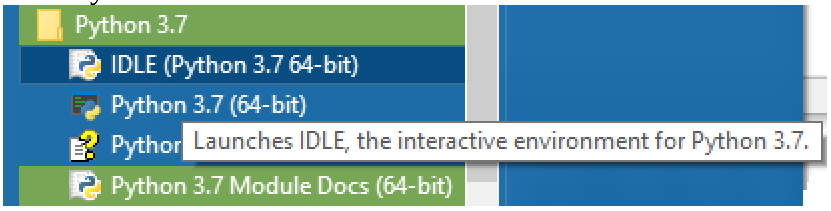
Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

New to Python? Start with the online tutorial and documentation.

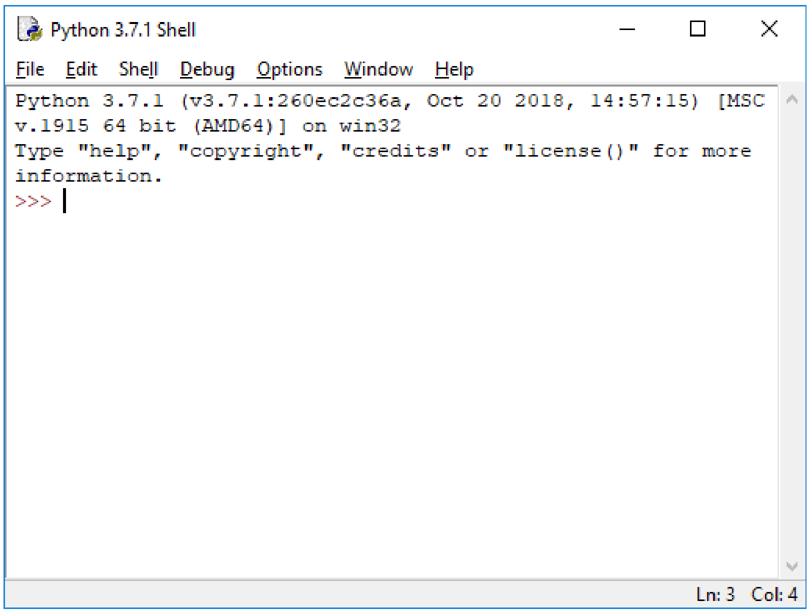
See what's new in this release.

Start Python: IDLE is the tool now.



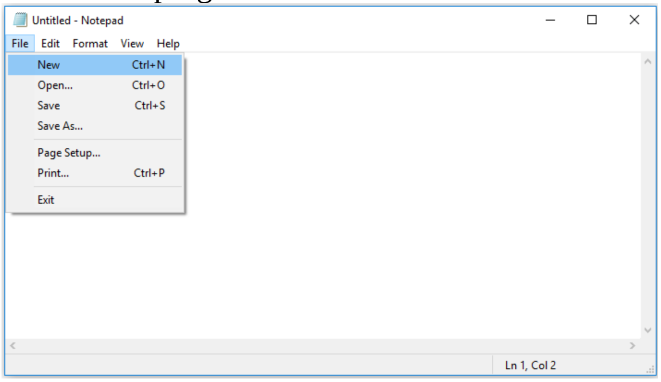
Python 3.7
IDLE (Python 3.7 64-bit)
Python 3.7 (64-bit)
Python
Python 3.7 Module Docs (64-bit)

Launches IDLE, the interactive environment for Python 3.7.



```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:57:15) [MSC
v.1915 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>> 
```

Start a new program

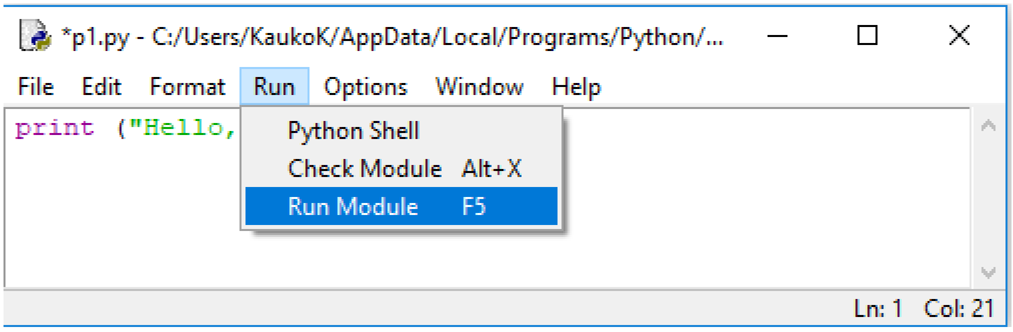


Save the new codefile and go on!

Let’s get to know a bit about Python tool!
Try first a pure console code:

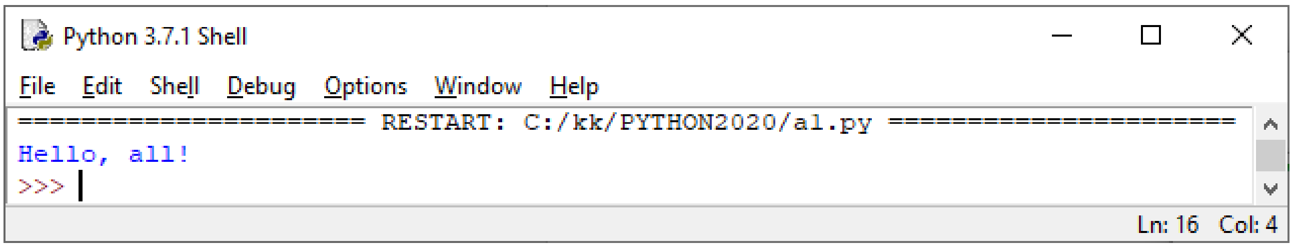
```
print(”Hello, all!”)
```

Then choose Run


*p1.py - C:/Users/KaukoK/AppData/Local/Programs/Python/...
File Edit Format Run Options Window Help
Python Shell
Check Module Alt+X
Run Module F5
Ln: 1 Col: 21

```
print ("Hello,
```

You get



```
======================= RESTART: C:/kk/PYTHON2020/a1.py =======================
Hello, all!
>>> 
```

Good!!

Let’s now start studying programming!!

Variables

Variables are storages used by the program. Memory for variables is allocated from computer's memory. That memory is called RAM (Random Access Memory) memory.

Variables have to be defined before they can be used.
In definition we need to tell data type and name of the variable.
Data type defines what kinds of values we can store to a variable:
Are they integers, e.g. values like 1, 20, 10000
Are they floating point (decimal) values, e.g. values like 2.35, 100.5555
Are they Characters, e.g. values like   'a', '4'
Are they boolean values, e.g. values like true, false
Are they texts (strings) , e.g. values like "Kokkola", "USA"

Python Data types names

Booleans
Numbers
Strings
Bytes
Lists
Tuples
Sets
Dictionaries

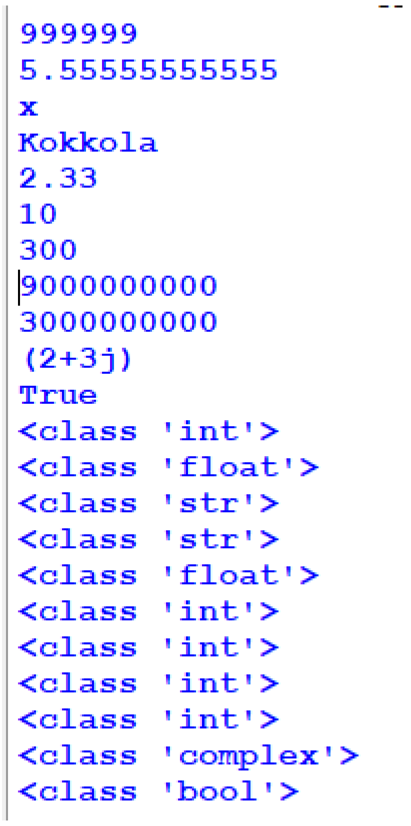
Number types are int, float, and complex
Boolean types is bool
String type is str

We concentrate on types numbers, bytes and strings and booleans.

Examples of data types (function type tells the datatype)

```
a = 999999
b = 5.55555555555
c = 'x'
d = "Kokkola"
e = 2.33
f = 10
g = 300
h = 9000000000
i = 3000000000
j = 2 + 3j
k = True


print(a)
print(b)
print(c)
print(d)
print(e)
print(f)
print(g)
print(h)
print(i)
print(j)
print(k)


print(type(a))
print(type(b))
print(type(c))
print(type(d))
print(type(e))
print(type(f))
print(type(g))
print(type(h))
print(type(i))
print(type(j))
print(type(k))
```

We get

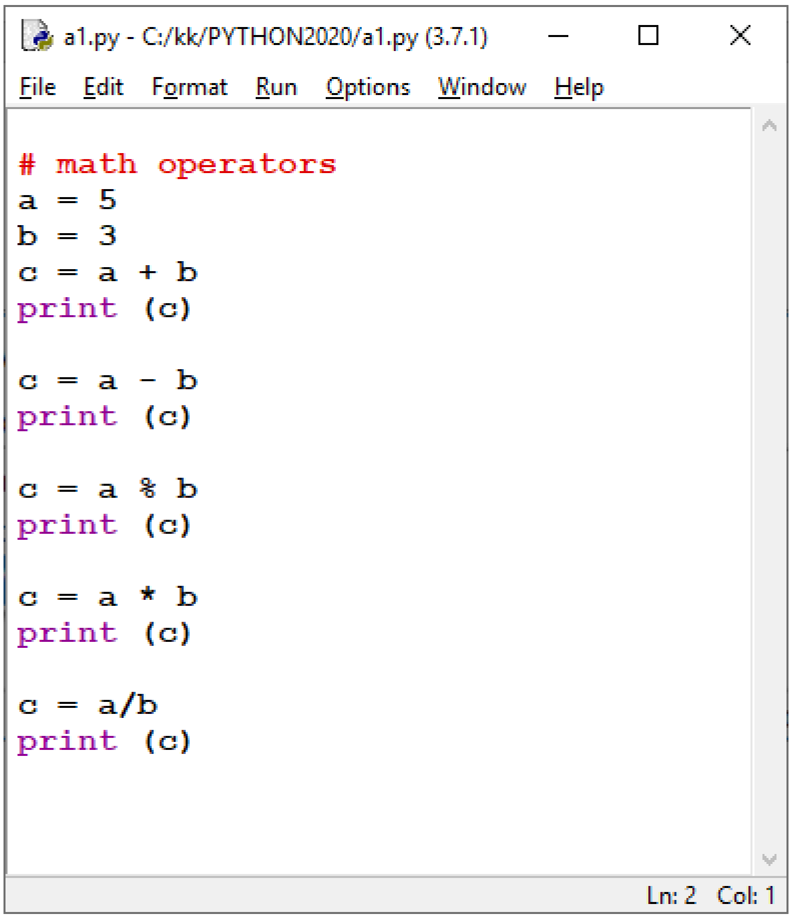


```
999999
5.55555555555
x
Kokkola
2.33
10
300
9000000000
3000000000
(2+3j)
True
<class 'int'>
<class 'float'>
<class 'str'>
<class 'str'>
<class 'float'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'complex'>
<class 'bool'>
```

Arithmetic operators

Operator	Explanation
+	Addition
-	Subtraction
*	Multiplication
%	Remainder
/	Division
//	Floor division
**	Exponentiation

Examples


a1.py - C:/kk/PYTHON2020/a1.py (3.7.1)
File Edit Format Run Options Window Help
Ln: 2 Col: 1

```
# math operators
a = 5
b = 3
c = a + b
print (c)

c = a - b
print (c)

c = a % b
print (c)

c = a * b
print (c)

c = a/b
print (c)
```

We get

```
8
2
2
15
1.6666666666666667
```

Math exercises

1
Create a mini calculator
2
Our programs uses Ohm's law to calculate the resistance.
User gives voltage and current.

3
User gives the speed of the car (km/h) and the distance (km). Program calculates amount of time.
a) in hours
b) in whole hours and minutes

4
Our program calculates BMI.

5
Create a euro converter: dollars to euros.

6
Convert seconds to hours, minutes, seconds.

7
Convert euros to 5, 10, 20, 50, 100, 200, 500 euros bills.

Decision making (branching)

Program flow is decided depending on the condition

Relational operators (to create conditions)

Operator	Explanation
<	Smaller than
<=	Smaller than or equal to
>	Bigger than
>=	Bigger than or equal to
==	Equal to
!=	Not equal to

if statement

Syntax:
```
if  this is true:
    this code is executed
```

Example
```
a = 5
if a != 5:
    print ("a is NOT 5!")
```

If several statements are executed after if, we use a program block that is created automatically with indents:
```
if  this is true :
    this code is executed
    and this code is executed
    and …
```

We can have else part, too:
```
if  this is true:
    this code is executed
else:
   this code is executed
```

Example of if else

```
a = 5
if  a != 5:
    print ("a is NOT 5!")
else:
    print ("a is  5!")
```

Example: program tells if given number is positive or not.

```
n = int(input("Give a whole number "))
if n >= 0:
   print ("is positive")
else:
   print("is negative")
```

We get

```
Give a whole number 4
is positive
```

If else exercises

1
User gives a value and our program tells if the value is > 100

2
Write a program which reads two integer values.
If the first is less than the second, print the message "up".
If the second is less than the first, print the message "down ".
If the numbers are equal, print the message "equal".

3
User enters a weekday number and the program tells the name of the day.

4
User gives a month number and our program tells the number of days in that month.

5
User gives the lengths of the triangle's sides. Program tells what is the triangle like and calculates the area of the triangle

Several choices-> several if-statements

```
n = int(input("Give a whole number "))

if n == 0:
    print ("zero")
elif n == 1:
    print ("one")
elif n == 2:
    print ("two")
else:
    print("other value")
```

We get

```
Give a whole number 2
two
```

Nested if-else statements

```
Example: is given number between 1 and?
x = 5
if x >= 1:
    if x <= 5:
        print("x is between 1 and 5")
    else:
        print("x is NOT between 1 and 5")
else:
    print("x is NOT between 1 and 5")
```

We get

```
x is between 1 and 5
```

Logical operators

Operator	Example
and	int a = 5; (a >= 0 && a <=10) true
or	(a < 0 || a > 10) false
not	

Example: is given value between 0 and 10?

```
// way 1

x = -3
if x >= 0:
    if x <= 10:
        print("x is between 0 and 10")
    else:
        print("x is NOT between 0 and 10")
else:
    print("x is NOT between 0 and 10")
```

```
// way 2

x = -3
if x >= 0 and x <= 10:
    print("x is between 0 and 10")
else:
    print("x is NOT between 0 and 10")
```

```
// way 3

x = -3
if x < 0 or x > 10:
    print("x is NOT between 0 and 10")
else:
    print("x is between 0 and 10")
```

Note: switch case is missing from Python

```
# Program prints the name of a value (between 0 and 5) in Italian.

n = 3
if n == 0:
    print("zero")
elif (n == 1):
    print("uno")
elif  n == 2:
    print("due");
elif  n == 3:
    print("tre");
elif n == 4:
    print("quattro");
elif  n == 5:
    print("cinque");
```

```
else:
    print("do not know");
```

```
We get
tre
```

Loops

`We use loops for` repeating some part of the code until some solution is found
A bit about program flow:
In programs execution flow can
a) go on straight forward (step by step)
b) contain decision making (branching)
c) contain loops

Examples of usages of loops:
when searching for a value from an array
when generating and printing hundreds of random numbers
in iterations

There are mainly two kinds of loops: for loop (when code is to be repeated fixed nnumber of time) and while loop (called often conditional loop).

for-loop

```
syntax
for definition:
  body of the loop

Program flow:
go straight forward (step by step)
decision making (branching)
loops
```

`Examples of using for loop`

```
Example: print out values 1 to 5
#print values 0 to 5
for x in range(6):
  print(x)


Example: print out 4, 8, 12, … 24
#print values 4, 8, ... 24
for x in range(4, 24, 4):
  print(x)

#Program calculates the sum of values 1 - 5
sum = 0
for x in range(6):
    sum = sum + x

print(sum)


#Program calculates the sum of even numbers between 2 - 40
sum = 0
for x in range(2,42, 2):
    sum = sum + x

print(sum)
#Program calculates sum: 5, 10, 15, .. 100.
sum = 0
```

```
for x in range(5, 105, 5):
    sum = sum + x

print(sum)
```

About random numbers

How to get random numbers?
Random object is needed: we have to import random module.
Import random
Then we can e.g. method
random.randint(lower limit, upper limit)
to get random values.

Getting values between 1 and 10:
x = random.randint(1,11)

Example: generate random numbers

```
import random
#Program generates 50 random numbers (between 1 to 10)

for x in range(50):
    y = random.randint(1,11)
    print (y)
```

Counting amounts of different numbers

```
n1 = 0
n2 = 0
n3 = 0
n4 = 0
n5 = 0

for x in range(50):
    y = random.randint(1,11)
    if y == 1:
        n1 += 1
    elif y == 2:
        n2 += 1
    elif y == 3:
        n3 += 1
    elif y == 4:
        n4 += 1
    elif y == 5:
        n5 += 1
print("Amounts:")
print(n1)
print(n2)
print(n3)
print(n4)
print(n5)
```

```
We get
Amounts:
5
7
3
```

```
3
4
```

Conditional loops

while loop

```
Syntax:
while (condition is true):
   code

Examples of while loop
#while
print ("while loop example")
k = 1
while k < 6:
    print(k)
    k = k + 1
```

Break and continue statements

Used with loops
Note: break was used even with switch-case
With break
you can terminate the loop when some condition is true
E.g.
When searching for a value from an array by using a loop:
when the value is found, there no use to go on searching,
just terminate the loop with break statement!
With continue
you can start a new round without executing the code that exists below continue statement
Example of using break:

```
#equation: 3x^3 - 2x^2 + 4x -7 = 0
x = -5.0
y = 0.0

while True:
    y = 3*x**3 - 2*x**2 + 4*x -7
    if y > -0.001 and y < 0.001:
        break
    x += 0.0001

print(x)
print(y)

We get
1.191400000001571
7.293985086676003e-05
```

Some special math operators

```
Assignment and math operators combined:
+=
-=
*=
/=
%=

Example:
x = 10;
x += 5  #  same as x = x + 5
```

Arrays

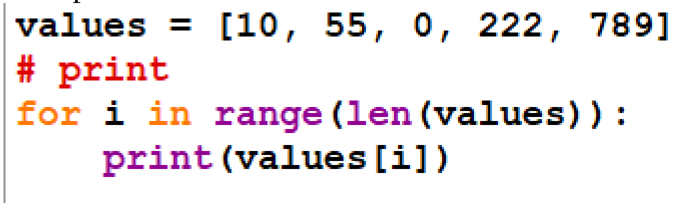
```
Normal variable can store only 1 value.
Array can store several values of same data type.
Note: Python has not real arrays, no built-in support for arrays, but we can use lists that
are almost similar than arrays in many other programming languages (
One dimensional array
Example: an array that can store 5 integers
```

10	55	0	222	789

```
values = [10, 55, 0, 222, 789]
Every place of an array has an index. The first index is 0.
Print (values[0])
Gives 10.

Python arrays have several methods that are used for handling array contents…
```

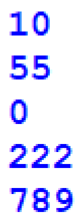
Method **len()** gives the size of an array.

Let's print all values



```
values = [10, 55, 0, 222, 789]
# print
for i in range(len(values)):
    print(values[i])
```

Result



```
10
55
0
222
789
```

2 dimensional array

Has rows and columns.
Example: Measures

1	2	3	4
122	132	99	96

measures = [[1,2,3,4],[122,132,99,96]]

Printing values

```
measures = [[1,2,3,4],[122,132,99,96]]
# print
for i in range(len(measures)):
    for j in range(len(measures[i])):
       print(measures[i][j], end='\t')
    print()
```

Result

```
1       2       3       4
122     132     99      96
```

How to initialize a 2 dimensional array?
Example here

```
personsAndSalaries = [["Bill", "2000"],["Ann", "2200"],
                      ["Tom", "3000"],   ["Jack", "1000"]]

for i in range(len(personsAndSalaries)):
                for j in range(len(personsAndSalaries[i])):
                               print(personsAndSalaries[i][j])
```

About string lists: in comparing you can use here common operators == and !=.
Here is an example

```
cities = ["Helsinki","Stockholm", "Oslo", "London"]
homeCity = "London"

for city in cities:
    if homeCity == city:
        print("It is in the list")
```

Result

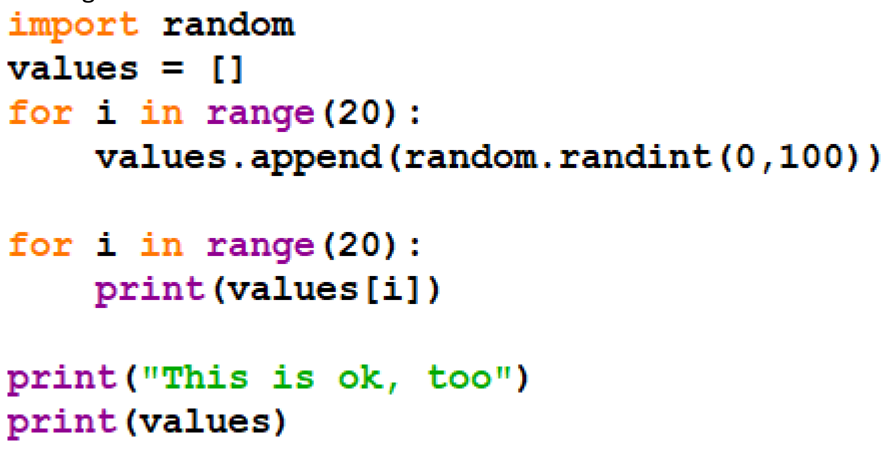
```
It is in the list
```

Basic array algorithms

Filling an array with random numbers
Calculate the sum and the average.

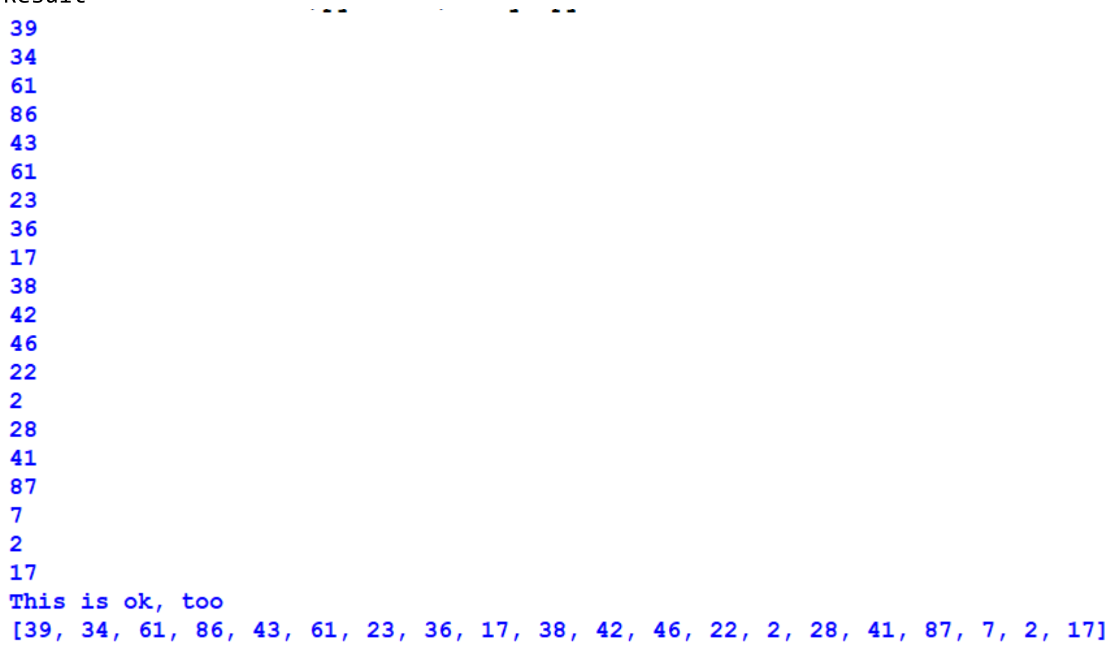
Searching for the minimun/maximum value
Checking if a specific value is in the table
Sorting an array

Filling with random numbers



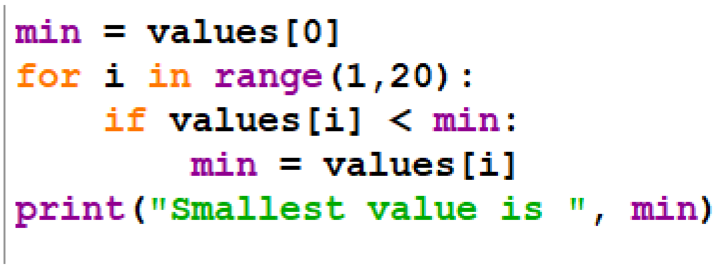
```
import random
values = []
for i in range(20):
    values.append(random.randint(0,100))

for i in range(20):
    print(values[i])

print("This is ok, too")
print(values)
```

Result



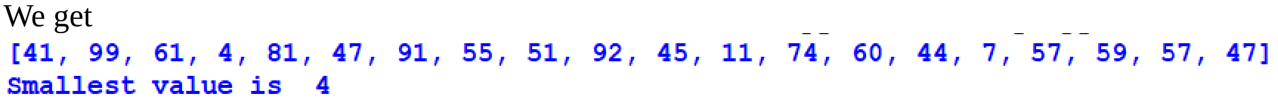
```
39
34
61
86
43
61
23
36
17
38
42
46
22
2
28
41
87
7
2
17
This is ok, too
[39, 34, 61, 86, 43, 61, 23, 36, 17, 38, 42, 46, 22, 2, 28, 41, 87, 7, 2, 17]
```

Minimun and maxmimun:
Principle:
Assign the first value of the table to some variable (e.g named min or max):
Then we check if there are smaller or bigger values in the remaining part of the array
If bigger or smaller value is found, it is assigned to min or max variable
Minimun:



```
min = values[0]
for i in range(1,20):
    if values[i] < min:
        min = values[i]
print("Smallest value is ", min)
```

We get

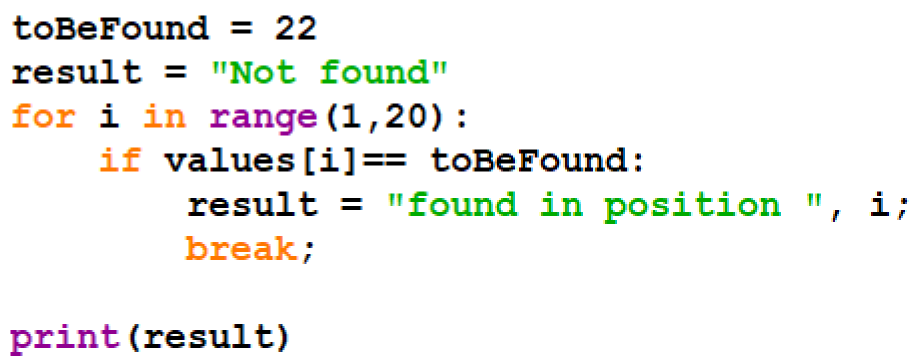


```
[41, 99, 61, 4, 81, 47, 91, 55, 51, 92, 45, 11, 74, 60, 44, 7, 57, 59, 57, 47]
Smallest value is  4
```

Maximun: this is one of the assignments!

Searching for a specific value

Principle:
Inside a loop we start comparing the value of array to the value we are searching for
If values are same
  add the position to some variable
  break the loop (no use to go on…)
After loop we can test the variable: if it has some positive value,
we can print that value was found
else
we print that it was not found
Code

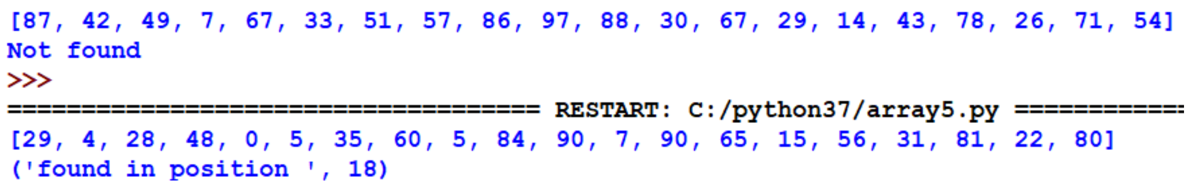


```
toBeFound = 22
result = "Not found"
for i in range(1,20):
    if values[i]== toBeFound:
        result = "found in position ", i;
        break;

print(result)
```

Result

1. run

2. run



```
[87, 42, 49, 7, 67, 33, 51, 57, 86, 97, 88, 30, 67, 29, 14, 43, 78, 26, 71, 54]
Not found
>>>
================================== RESTART: C:/python37/array5.py ===========
[29, 4, 28, 48, 0, 5, 35, 60, 5, 84, 90, 7, 90, 65, 15, 56, 31, 81, 22, 80]
('found in position ', 18)
```

Sorting: We use here selection sort method (slow method, but good for demonstration)

Example table:

6	7	3	9	2	99

First we compare the first value to others value and swap values when needed:
1. round:
7 < 6? no
3 < 6? yes, swap

3	7	6	9	2	99

9 < 3? no
2 < 3? yes, swap

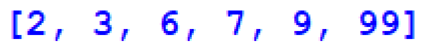
2	7	6	9	3	99

99 < 2? no

Code

```
import random
values = [3,7,6,9,2,99]
for i in range(6):
    for j in range(i+1,6):
        if values[i] > values[j]:
            temp = values[i]
            values[i] = values[j]
            values[j] = temp

print(values)
```

Result



```
[2, 3, 6, 7, 9, 99]
```

Contents

Functions

Introduction to functions

Functions are often called also subprograms, routines, procedures, methods…
Functions
do one well defined task
Instead of putting all the code the main body of the program, we can use
functions and call them when needed.

Why functions?
Can be called several times from other parts of the program
Can be reused in other programs
Program is better organized (better structure)
No need for repeating same code
SO, When some code is to be used more than once, it is good to create a function

Function definition (implementation)

def functionName(parameters):
    function body (the code, implementation)

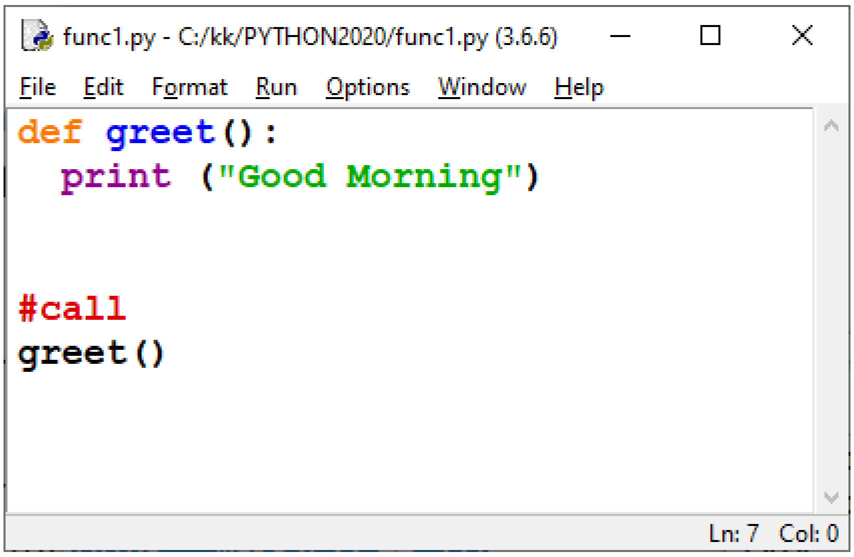
Learning by Examples

Example 1
Our function prints out “Good Morning”

def greet():
  print (“Good Morning”)

Function call
greet()

Test run



func1.py - C:/kk/PYTHON2020/func1.py (3.6.6)
File Edit Format Run Options Window Help
Ln: 7 Col: 0

```
def greet():
   print ("Good Morning")


#call
greet()
```

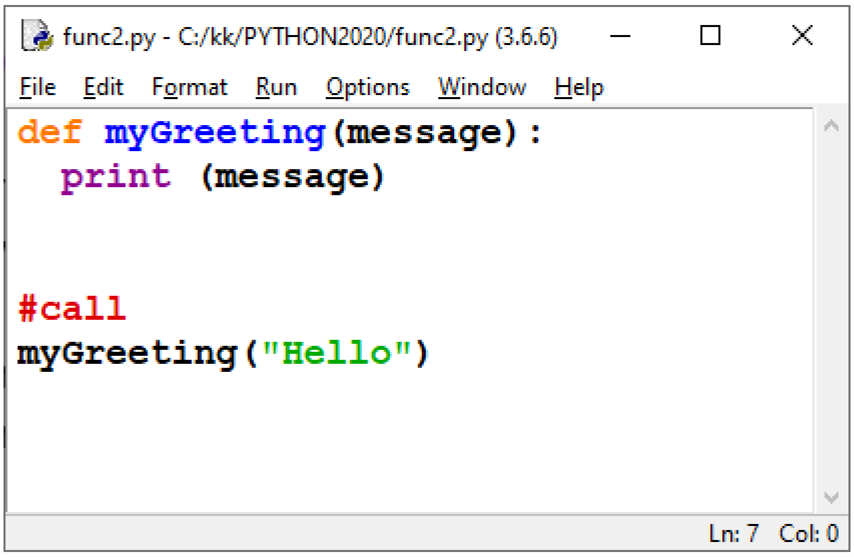
Result

```
Good Morning
```

Example 2
We want do decide ourselves what to print!

def myGreeting(message):
  print (message)

Test run


func2.py - C:/kk/PYTHON2020/func2.py (3.6.6)
File Edit Format Run Options Window Help
Ln: 7 Col: 0

```
def myGreeting(message):
  print (message)


#call
myGreeting("Hello")
```

Result:

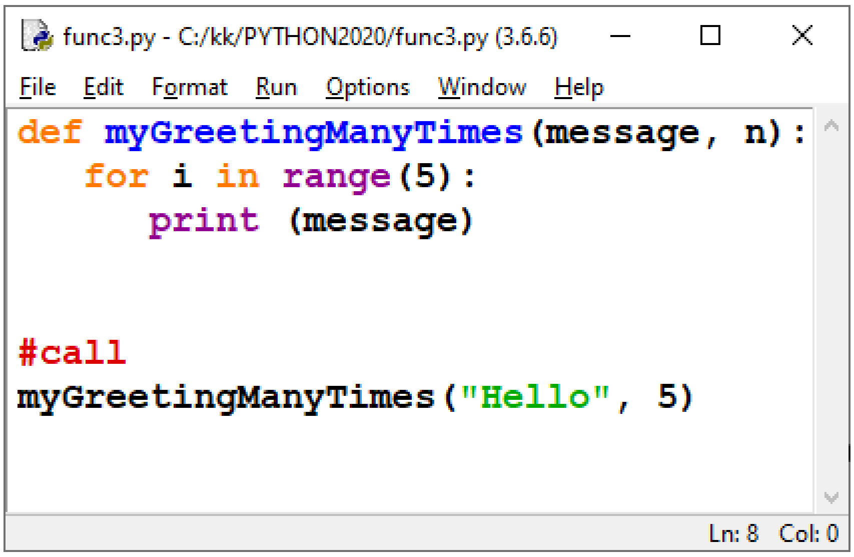
```
Hello
```

Example 3
We have had only 1 parameter – let's try with different kinds of function parameters now...

Example 4
Our function prints our greeting n times.

```
def myGreetingManyTimes(message, n):
   for i in range(5):
      print (message)
```

Test run


func3.py - C:/kk/PYTHON2020/func3.py (3.6.6)
File Edit Format Run Options Window Help
Ln: 8 Col: 0

```
def myGreetingManyTimes(message, n):
   for i in range(5):
      print (message)


#call
myGreetingManyTimes("Hello", 5)
```

Result

```
Hello
Hello
Hello
Hello
Hello
```

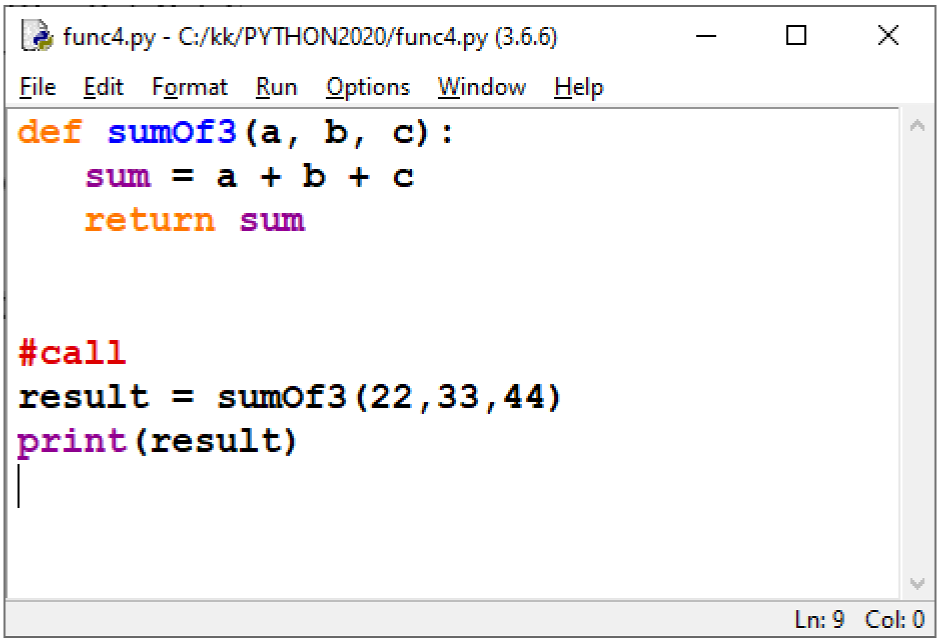
Functions can also return values. Let's take a look at that feature.

Example 5
Next function returns the sum of 3 whole numbers.

```
def sumOf3(a, b, c):
  sum = a + b + c
  return sum
```

Note, we have return statement there!

Test run


func4.py - C:/kk/PYTHON2020/func4.py (3.6.6)
File Edit Format Run Options Window Help
Ln: 9 Col: 0

```
def sumOf3(a, b, c):
    sum = a + b + c
    return sum


#call
result = sumOf3(22,33,44)
print(result)
```

Result

```
99
```

We can call that function also like this:

print (sumOf3(55,66,88))

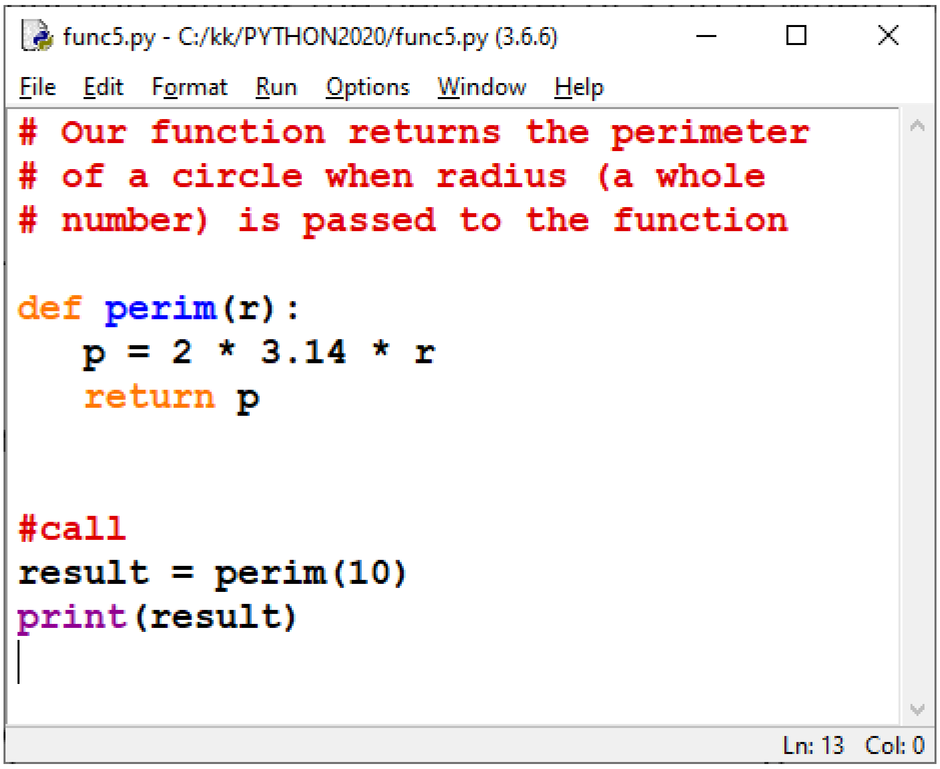
OR

```
x = 10
y = 20
z = 20

print (sumOf3(x, y, z))
```

Example 6
Our function returns the perimeter of a circle when radius (a whole number) is passed to the function

```
def perim(r):
  p = 2 * 3.14 * r
  return p
```

Test run


func5.py - C:/kk/PYTHON2020/func5.py (3.6.6)
File Edit Format Run Options Window Help
Ln: 13 Col: 0

```
# Our function returns the perimeter
# of a circle when radius (a whole
# number) is passed to the function

def perim(r):
   p = 2 * 3.14 * r
   return p


#call
result = perim(10)
print(result)
```

Result

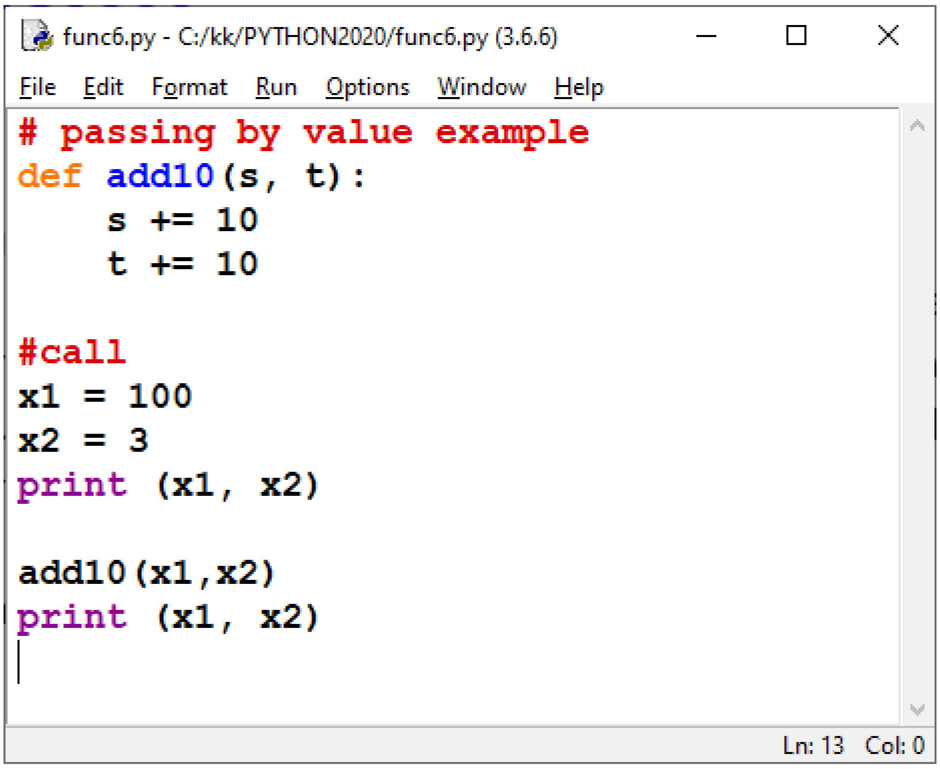
```
62.80000
```

Note: we could have taken the value of pi from math, but libraries are discussed later...

Passing by value or passing by reference

Here is an example where function has normal parameters. Values of are modified inside the function but
original variables are not changed. This is called passing by value: only the value of the variable is passed to the function and original variable cannot be modified by the function (function does not an access to original memory place).

Example 7


func6.py - C:/kk/PYTHON2020/func6.py (3.6.6)
File Edit Format Run Options Window Help
Ln: 13 Col: 0

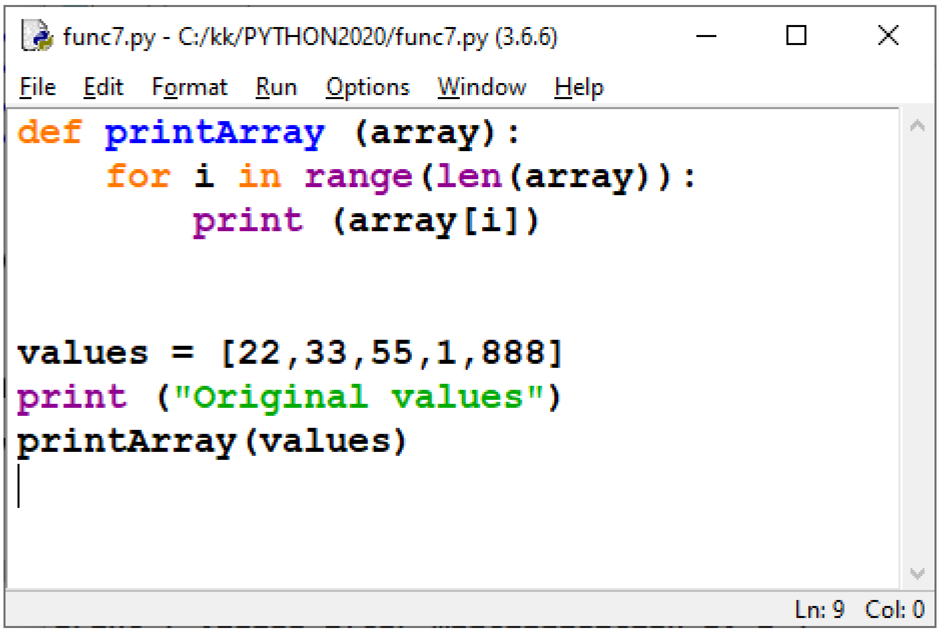
```
# passing by value example
def add10(s, t):
    s += 10
    t += 10

#call
x1 = 100
x2 = 3
print (x1, x2)

add10(x1,x2)
print (x1, x2)
```

Test run shows that values of x1 and x2 are not changed:

```
100 3
100 3
```

Arrays are passed to functions as references – they can be modified by the function.

Example 7
First we only print an array


func7.py - C:/kk/PYTHON2020/func7.py (3.6.6)
File Edit Format Run Options Window Help
Ln: 9 Col: 0

```
def printArray (array):
    for i in range(len(array)):
        print (array[i])


values = [22,33,55,1,888]
print ("Original values")
printArray(values)
```

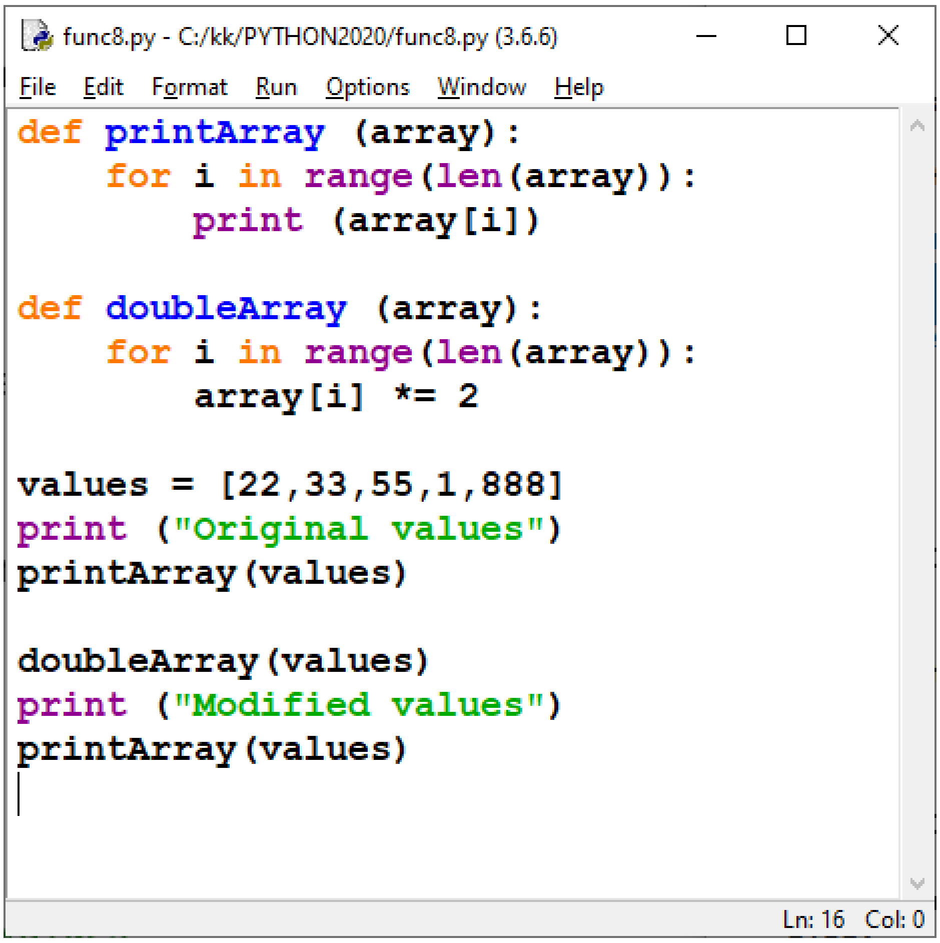
Result

```
Original values
22
33
55
1
888
```

Example 8
Here arrays values are multiplied by 2. You can see that original array has changed.

```
def doubleArray (array):
    for i in range(len(array)):
        array[i] *= 2
```

Test run


func8.py - C:/kk/PYTHON2020/func8.py (3.6.6)
File Edit Format Run Options Window Help
Ln: 16 Col: 0

```python
def printArray (array):
    for i in range(len(array)):
        print (array[i])

def doubleArray (array):
    for i in range(len(array)):
        array[i] *= 2

values = [22,33,55,1,888]
print ("Original values")
printArray(values)

doubleArray(values)
print ("Modified values")
printArray(values)
```

AND

```
Original values
22
33
55
1
888
Modified values
44
66
110
2
1776
```

We return to functions even later.
This was part one...