Notes Sanna Heikura

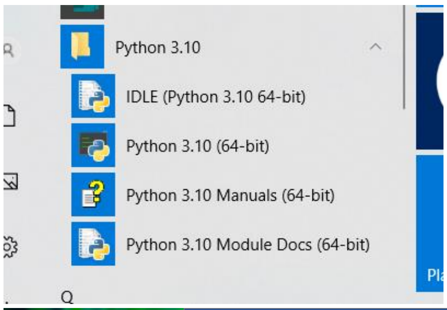
Part 1

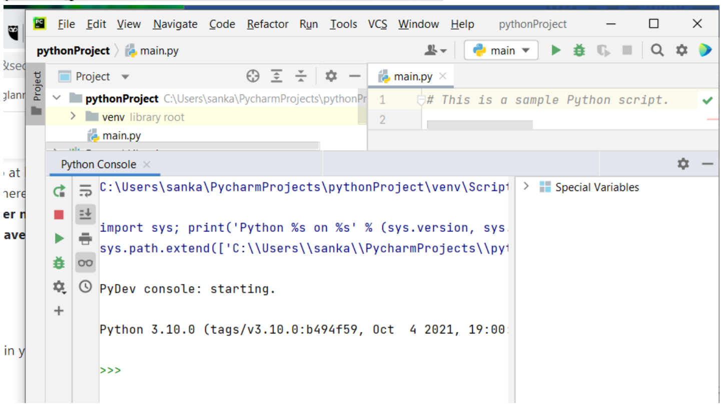
1. **What is Phyton & History**
   Phyton was developed by Guido van Rossum in the late 80ies and early 90ies in the Netherlands. It is the most popular programming language and used as an introduction language. It can be used to control website servers, data analytics and more.

   Phyton is interpreted that means that you have a program Phyton IDE that runs the code that you create.

2. **Installing Phyton and and IDE**

   I have Phyton and Phycharm installed on my computer.

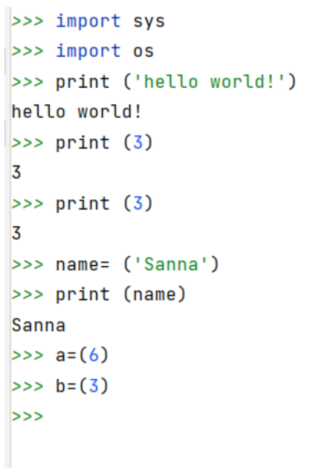

Python 3.10
IDLE (Python 3.10 64-bit)
Python 3.10 (64-bit)
Python 3.10 Manuals (64-bit)
Python 3.10 Module Docs (64-bit)


File Edit View Navigate Code Refactor Run Tools VCS Window Help pythonProject
pythonProject main.py
main
Project
pythonProject C:\Users\sanka\PycharmProjects\pythonPr
venv library root
main.py
Python Console
Special Variables

```
# This is a sample Python script.
```

```
C:\Users\sanka\PycharmProjects\pythonProject\venv\Scripi

import sys; print('Python %s on %s' % (sys.version, sys
sys.path.extend(['C:\\Users\\sanka\\PycharmProjects\\py

PyDev console: starting.

Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00

>>>
```

3. **Hello World**
   Hello world + variables examples

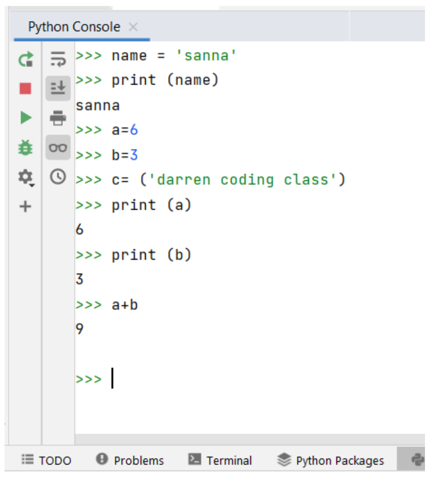


```
>>> import sys
>>> import os
>>> print ('hello world!')
hello world!
>>> print (3)
3
>>> print (3)
3
>>> name= ('Sanna')
>>> print (name)
Sanna
>>> a=(6)
>>> b=(3)
>>>
```

4. Variables

- *Variables must always start with a letter and then can include numbers and an underscores eg: N34456_78_six*
- *Variables in Python can contain the same string or values eg: b = 2, c = 2*

Examples in Phycharm:

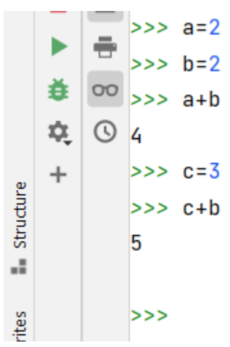


```
>>> name = 'sanna'
>>> print (name)
sanna
>>> a=6
>>> b=3
>>> c= ('darren coding class')
>>> print (a)
6
>>> print (b)
3
>>> a+b
9

>>> |
```

5. Swapping variables

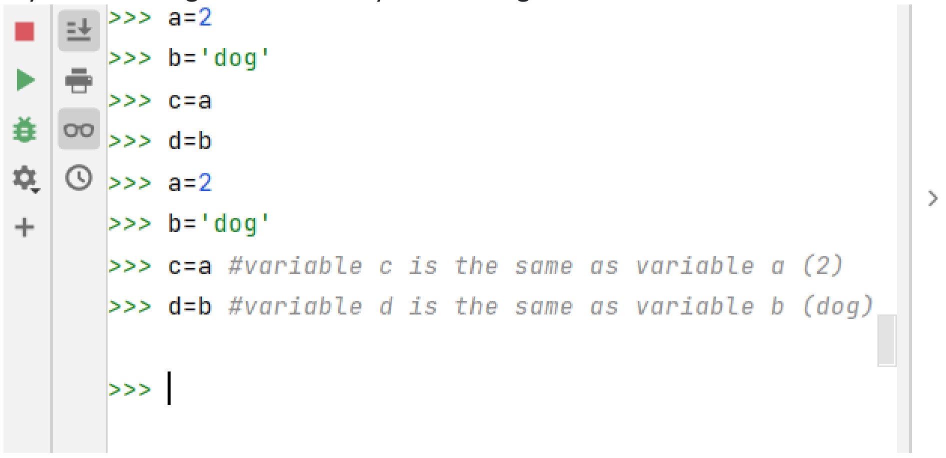
Variable strings or values can be switched at any time or changed.

Examples:



```
>>> a=2
>>> b=2
>>> a+b
4
>>> c=3
>>> c+b
5

>>>
```

If you have assigned a variable you can assign other variables to the same value or string.

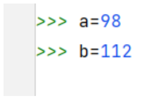


```
>>> a=2
>>> b='dog'
>>> c=a
>>> d=b
>>> a=2
>>> b='dog'
>>> c=a #variable c is the same as variable a (2)
>>> d=b #variable d is the same as variable b (dog)

>>> |
```

6. Variable types

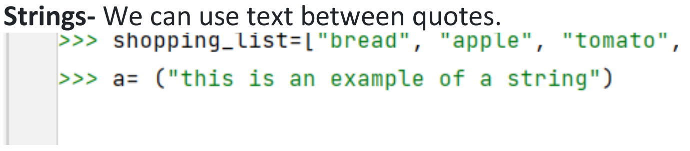
There are five types of variables:

Numbers- You can use numbers as variables. We need to use brackets to calculate parts of an a formula that we want to go first.



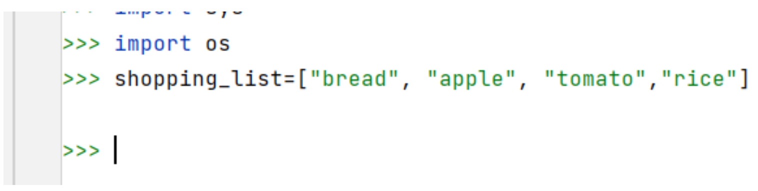
```
>>> a=98
>>> b=112
```

Strings- We can use text between quotes.



```
>>> shopping_list=["bread", "apple", "tomato",
>>> a= ("this is an example of a string")
```

Lists- We can create lists of things, for instance a shopping list. List use the square brackets [ ]. The items in the list are separated with a comma.



```
>>> import os
>>> shopping_list=["bread", "apple", "tomato","rice"]

>>> |
```

How does this differ from an array? Or does it?

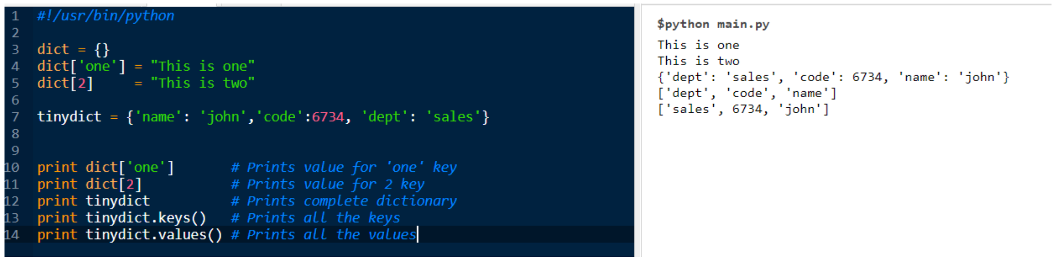
Tuples-Tuple is lists are similar to lists. A tuple consists of a number of values separated by commas.

```
>>> tuple= ["milk",10 "Tomatoes",5 "bread",25 "Pasta", 12]
...
```

10 milks, 5 tomatos, 25 breads, 12 pastas …..

Dictionaries- They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([])



```
#!/usr/bin/python

dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}


print dict['one']       # Prints value for 'one' key
print dict[2]           # Prints value for 2 key
print tinydict          # Prints complete dictionary
print tinydict.keys()   # Prints all the keys
print tinydict.values() # Prints all the values
```

```
$python main.py
This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
['dept', 'code', 'name']
['sales', 6734, 'john']
```

7. Variable arithmetic operators

There are 7 arimethric operators. We need to use brackets to calculate parts of a formula we want to calculate first!

Examples:

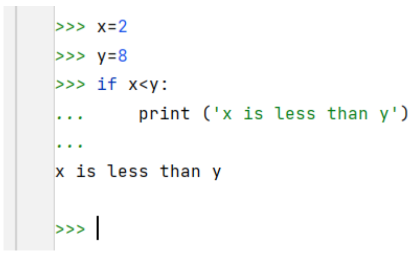
```
>>> a = 6
>>> b = 3
>>> d = 10
>>> e = 15
>>> (a+b) / 3
3.0
>>> (a+b) / 2 + (e * a)
94.5

>>>
```

Part 2

1. if clause

   The if statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison. If the boolean expression evaluates to TRUE, then the block of statement(s) inside the if statement is executed. If boolean expression evaluates to FALSE, then the first set of code after the end of the if statement(s) is executed.

   Example:



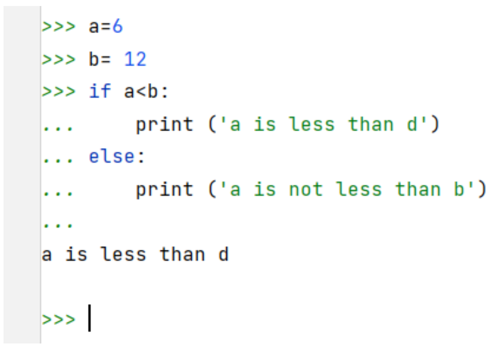
```
>>> x=2
>>> y=8
>>> if x<y:
...     print ('x is less than y')
...
x is less than y

>>> |
```

2. else

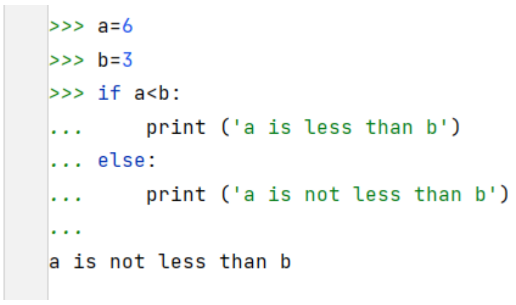
   An else statement can be combined with an if statement. An else statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

   The else statement is an optional statement and there could be at most only one else statement following if.

   Example:



```
>>> a=6
>>> b= 12
>>> if a<b:
...     print ('a is less than d')
... else:
...     print ('a is not less than b')
...
a is less than d

>>> |
```



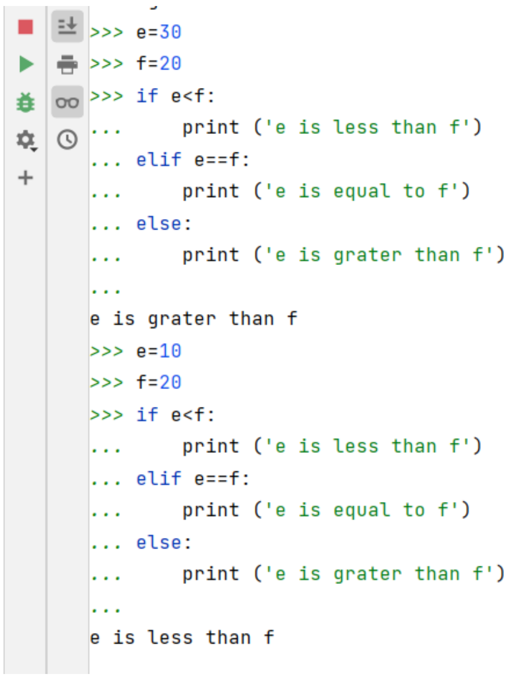
```
>>> a=6
>>> b=3
>>> if a<b:
...     print ('a is less than b')
... else:
...     print ('a is not less than b')
...
a is not less than b
```

3. equal to (==)

Double equal signs mean equal to the value and not a declaration of what the value should be.

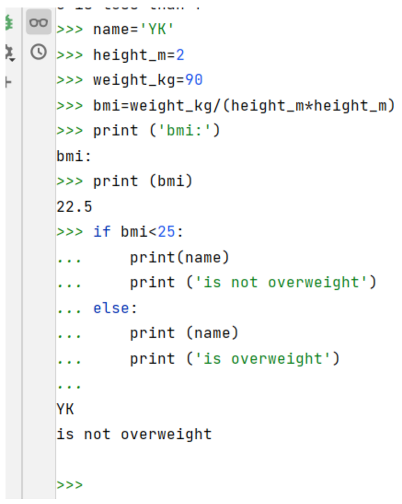
elif statement lets you check many expressions for TRUE and run a block of code as soon as one of the conditions is TRUE. Like the else, the **elif statement** is optional.

Example:

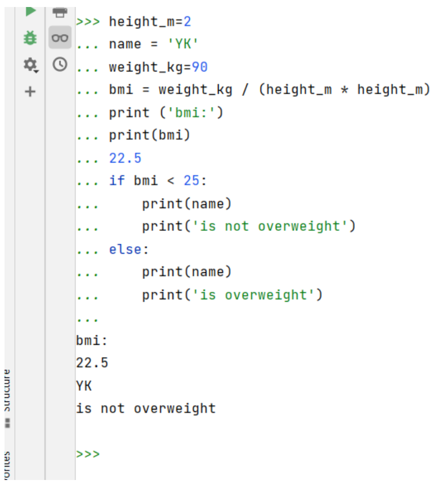


```
>>> e=30
>>> f=20
>>> if e<f:
...     print ('e is less than f')
... elif e==f:
...     print ('e is equal to f')
... else:
...     print ('e is grater than f')
...
e is grater than f
>>> e=10
>>> f=20
>>> if e<f:
...     print ('e is less than f')
... elif e==f:
...     print ('e is equal to f')
... else:
...     print ('e is grater than f')
...
e is less than f
```

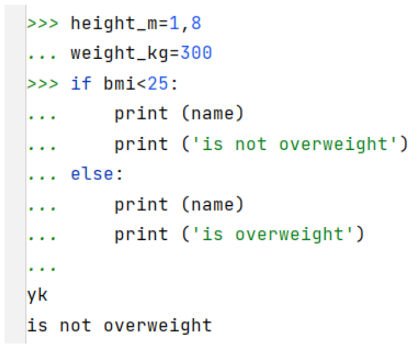
4. BMI calculator with different values



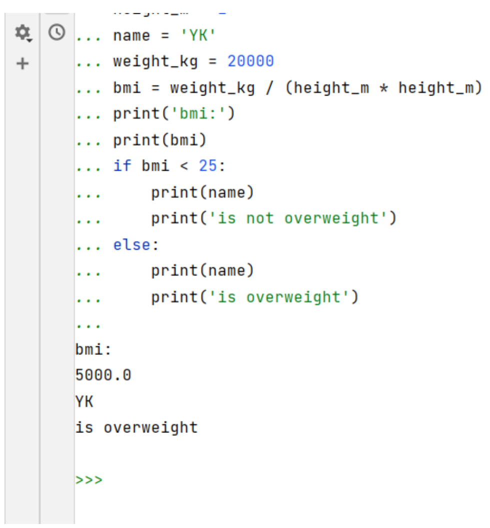
```
>>> name='YK'
>>> height_m=2
>>> weight_kg=90
>>> bmi=weight_kg/(height_m*height_m)
>>> print ('bmi:')
bmi:
>>> print (bmi)
22.5
>>> if bmi<25:
...     print(name)
...     print ('is not overweight')
... else:
...     print (name)
...     print ('is overweight')
...
YK
is not overweight

>>>
```



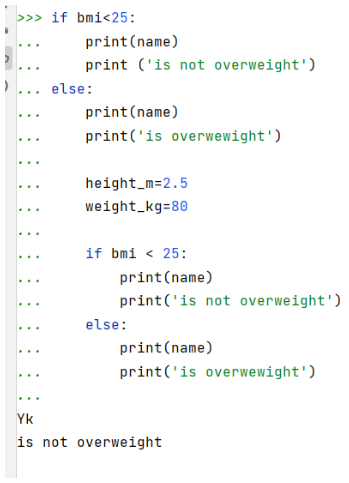
```
>>> height_m=2
... name = 'YK'
... weight_kg=90
... bmi = weight_kg / (height_m * height_m)
... print ('bmi:')
... print(bmi)
... 22.5
... if bmi < 25:
...     print(name)
...     print('is not overweight')
... else:
...     print(name)
...     print('is overweight')
...
bmi:
22.5
YK
is not overweight

>>>
```



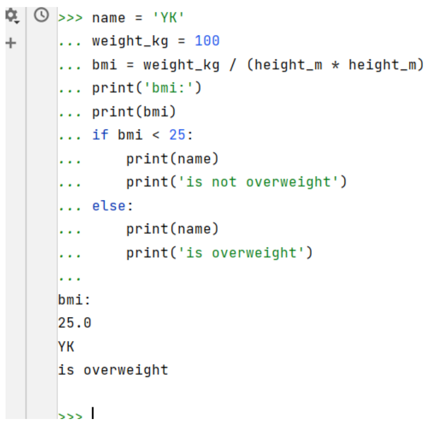
```
>>> height_m=1,8
... weight_kg=300
>>> if bmi<25:
...     print (name)
...     print ('is not overweight')
... else:
...     print (name)
...     print ('is overweight')
...
yk
is not overweight
```



```
... name = 'YK'
... weight_kg = 20000
... bmi = weight_kg / (height_m * height_m)
... print('bmi:')
... print(bmi)
... if bmi < 25:
...     print(name)
...     print('is not overweight')
... else:
...     print(name)
...     print('is overweight')
...
bmi:
5000.0
YK
is overweight

>>>
```



```
>>> if bmi<25:
...     print(name)
...     print ('is not overweight')
... else:
...     print(name)
...     print('is overwewight')
...
...     height_m=2.5
...     weight_kg=80
...
...     if bmi < 25:
...         print(name)
...         print('is not overweight')
...     else:
...         print(name)
...         print('is overwewight')
...
Yk
is not overweight
```



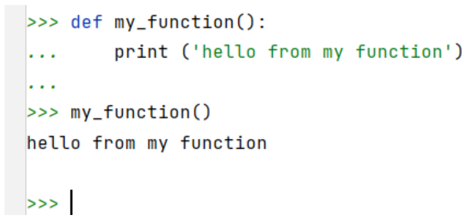
```
>>> name = 'YK'
... weight_kg = 100
... bmi = weight_kg / (height_m * height_m)
... print('bmi:')
... print(bmi)
... if bmi < 25:
...     print(name)
...     print('is not overweight')
... else:
...     print(name)
...     print('is overweight')
...
bmi:
25.0
YK
is overweight

>>> |
```

5. Functions

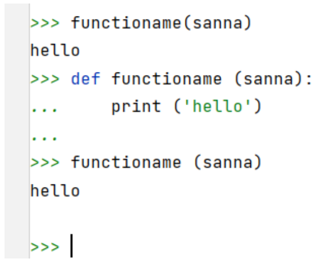
One way to think about functions is that it´s a collection of instructions or a collection of code.

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result. In phyton a function is defined using the def keyword. To call a function, use the function name followed by parenthesis.

Example:

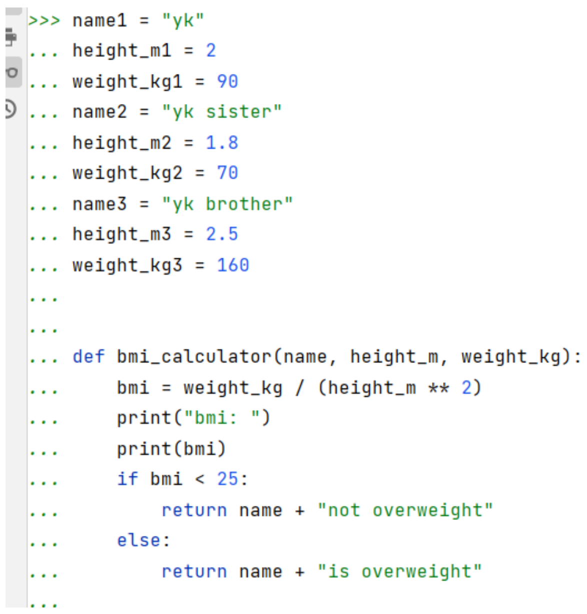


```
>>> def my_function():
...     print ('hello from my function')
...
>>> my_function()
hello from my function

>>> |
```

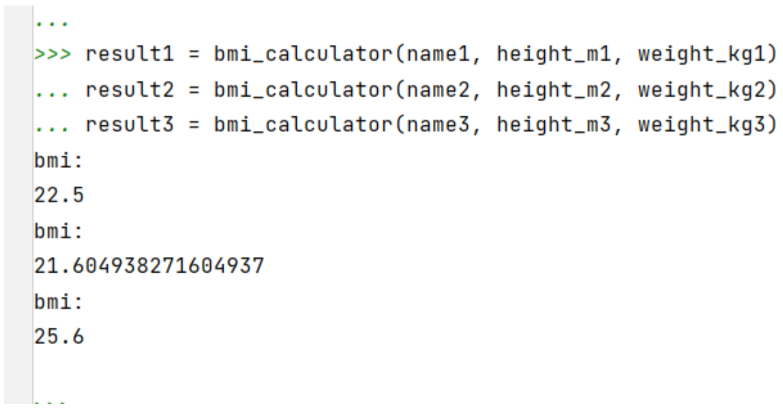


```
>>> functioname(sanna)
hello
>>> def functioname (sanna):
...     print ('hello')
...
>>> functioname (sanna)
hello

>>> |
```

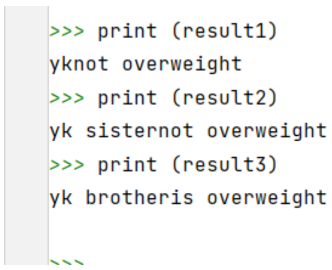
6. BMI calculator (using functions)



```
>>> name1 = "yk"
... height_m1 = 2
... weight_kg1 = 90
... name2 = "yk sister"
... height_m2 = 1.8
... weight_kg2 = 70
... name3 = "yk brother"
... height_m3 = 2.5
... weight_kg3 = 160
...
...
... def bmi_calculator(name, height_m, weight_kg):
...     bmi = weight_kg / (height_m ** 2)
...     print("bmi: ")
...     print(bmi)
...     if bmi < 25:
...         return name + "not overweight"
...     else:
...         return name + "is overweight"
...
```



```
...
>>> result1 = bmi_calculator(name1, height_m1, weight_kg1)
... result2 = bmi_calculator(name2, height_m2, weight_kg2)
... result3 = bmi_calculator(name3, height_m3, weight_kg3)
bmi:
22.5
bmi:
21.604938271604937
bmi:
25.6
```

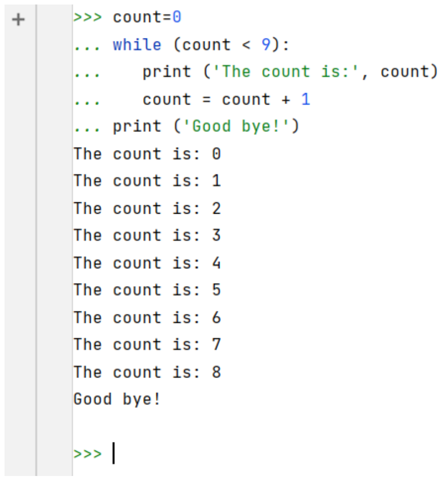


```
>>> print (result1)
yknot overweight
>>> print (result2)
yk sisternot overweight
>>> print (result3)
yk brotheris overweight
```

Part 3

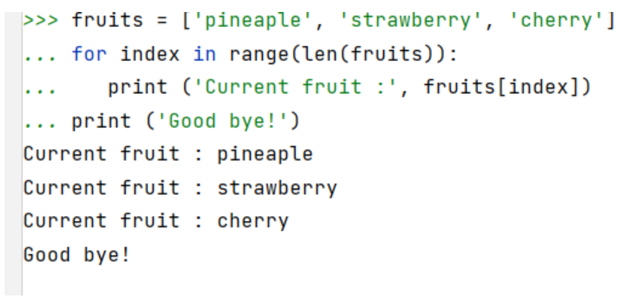
1. **Loops**

While Loop - Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.



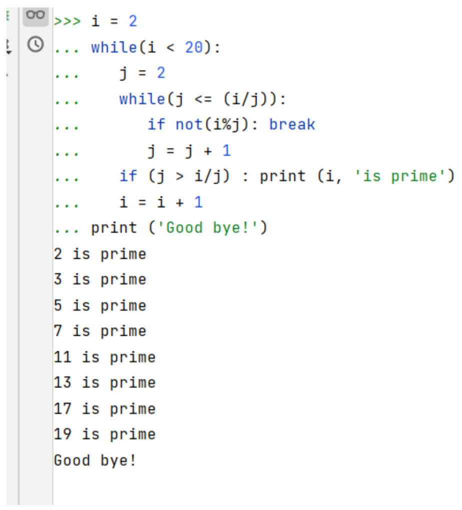
```
>>> count=0
... while (count < 9):
...     print ('The count is:', count)
...     count = count + 1
... print ('Good bye!')
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!

>>> |
```

For loops- Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.



```
>>> fruits = ['pineaple', 'strawberry', 'cherry']
... for index in range(len(fruits)):
...     print ('Current fruit :', fruits[index])
... print ('Good bye!')
Current fruit : pineaple
Current fruit : strawberry
Current fruit : cherry
Good bye!
```

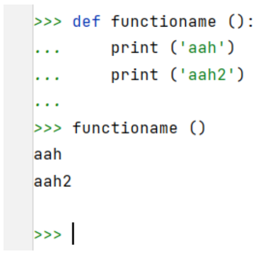
Nested loops- it means you can use one or more loop inside any other while or for loop



```
>>> i = 2
... while(i < 20):
...    j = 2
...    while(j <= (i/j)):
...       if not(i%j): break
...       j = j + 1
...    if (j > i/j) : print (i, 'is prime')
...    i = i + 1
... print ('Good bye!')
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
Good bye!
```

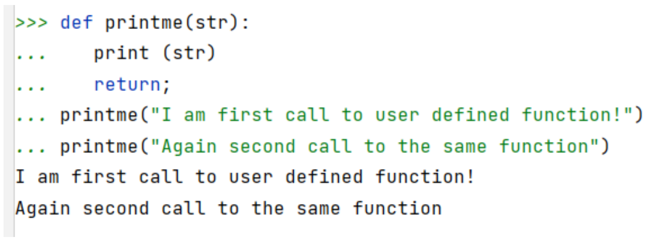
2. Functions

Creating a function: As described earlier function blocks begin with a def keyword. After that you write the function name and parentheses (). You put the argument in the brackets. After that you put a colon: after the (). Then you write your statements. The statement returns when called with the functioname.



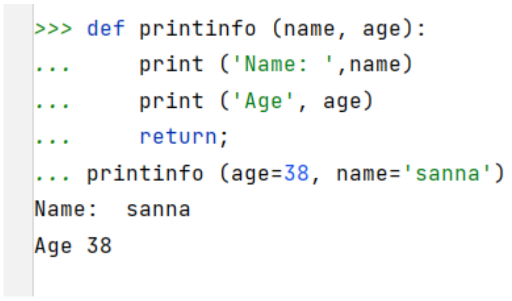
```
>>> def functioname ():
...     print ('aah')
...     print ('aah2')
...
>>> functioname ()
aah
aah2

>>> |
```

Printme() function



```
>>> def printme(str):
...     print (str)
...     return;
... printme("I am first call to user defined function!")
... printme("Again second call to the same function")
I am first call to user defined function!
Again second call to the same function
```

Keywords argument function

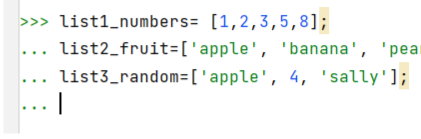


```
>>> def printinfo (name, age):
...     print ('Name: ',name)
...     print ('Age', age)
...     return;
... printinfo (age=38, name='sanna')
Name:  sanna
Age 38
```

3. Lists

List are used to block stuff into one same line of code. Much like the arrays in Java script. For instance we can create list of numbers or fruits or any other categories we choose. The items do not need to be the same type.

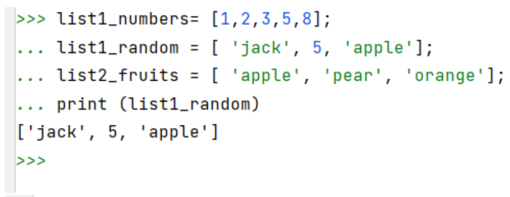
Creating lists:



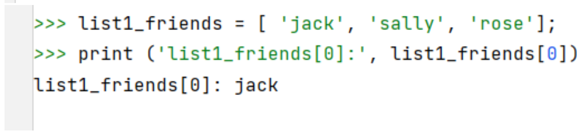
```
>>> list1_numbers= [1,2,3,5,8];
... list2_fruit=['apple', 'banana', 'pear']
... list3_random=['apple', 4, 'sally'];
... |
```

Accessing list and values in a list:

The number zero has to be used to count from the beginning of the list. Number one will be counted from the second item and so on.



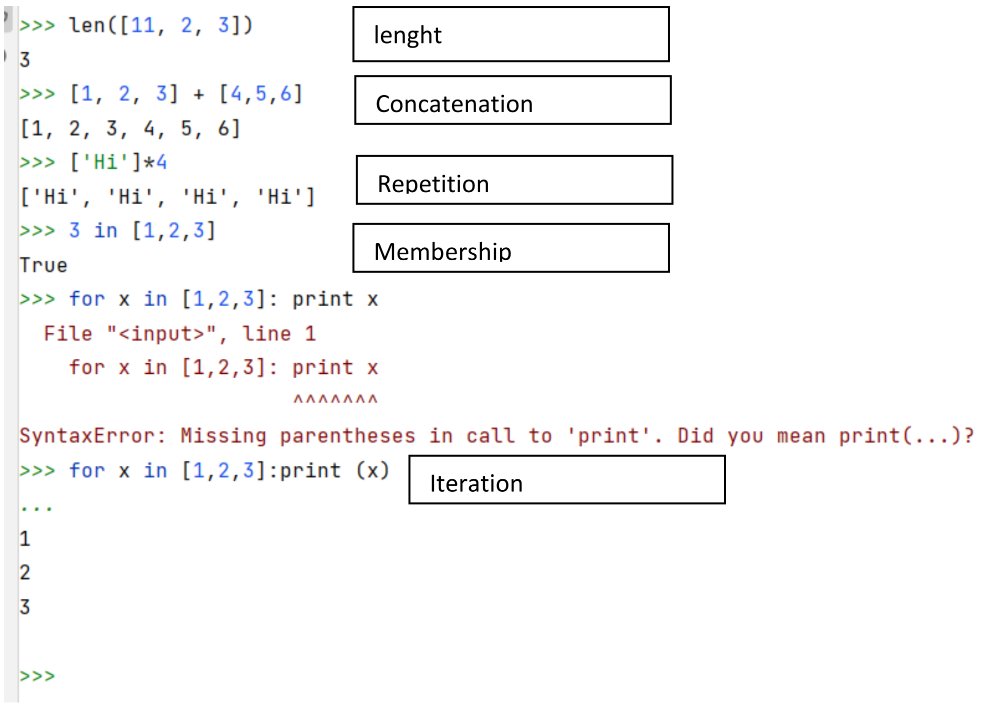
```
>>> list1_numbers= [1,2,3,5,8];
... list1_random = [ 'jack', 5, 'apple'];
... list2_fruits = [ 'apple', 'pear', 'orange'];
... print (list1_random)
['jack', 5, 'apple']
>>>
```



```
>>> list1_friends = [ 'jack', 'sally', 'rose'];
>>> print ('list1_friends[0]:', list1_friends[0])
list1_friends[0]: jack
```

Basic List operators:

Lists respond to the + and * operators like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.



```
>>> len([11, 2, 3])
3
>>> [1, 2, 3] + [4,5,6]
[1, 2, 3, 4, 5, 6]
>>> ['Hi']*4
['Hi', 'Hi', 'Hi', 'Hi']
>>> 3 in [1,2,3]
True
>>> for x in [1,2,3]: print x
  File "<input>", line 1
    for x in [1,2,3]: print x
                      ^^^^^^^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
>>> for x in [1,2,3]:print (x)
...
1
2
3

>>>
```

lenght

Concatenation

Repetition

Membership

Iteration