DevOps Course Summary

Lesson 2

Statements

https://docs.python.org/3.6/reference/compound_stmts.html

Loops:

• A loop is a sequence of statements that is specified once but which may be carried out several times in succession.

• The code "inside" the loop (the body of the loop) is obeyed a specified number of times, or once for each of a collection of items, or until some condition is met, or indefinitely.

• In Python, we have a few loops types:
- o For o
- While

For loop:

• Use for loop when the number of the iterations is known before entering into the body of the loop.

• It has the flexibility to assign variables before entering into the body of the loop and perform an update at the end of the loop.

• Syntax contains 3 parameters:
o Variable initialization (x).
o Condition (range 5).
o Statement (printing).

• To put the code above in simple words, will be: ***Increment X value by 1 in each iteration, and keep running as long as X is smaller than 5.***

```
for x in range(5):
    print(x)
```

• For loop has a few variations:

o Run for X times:

```
for x in range(5):
    print(x)
```

```
Result: 0 1 2 3 4
```

o Run from a specific index (3) X times (5):

```
for x in range(3,5):
    print(x)
```

```
Result: 3 4
```

o Run from a specific index X times, but increment X in 2 every iteration:

```
for x in range(3,8,2):
    print(x)
```

```
Result: 3 5 7
```

<u>While loop:</u>

• The block of statements will be repeated as long as the condition returns true.

• The condition should return false for exiting the loop.

• The 'while' loop can be used when the number of iterations is unknown.

```
while condition:
Block to execute
```

```
count = 0
while count < 5:
    print(count)
    count += 1
```

<u>Break statement</u>

When a break statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop.

```
count = 0
while 1 > 0:
    print(count)
    count += 1
    if count >= 5:
        break
```

Continue statement

Continue statement forces an early iteration of the loop.

```
for x in range(5):
    if x == 3:
        continue
    print(x)
```

Modules (files)

• In Python, one file is called a module.

• A module can consist of multiple classes or functions.

• As Python is not an OO language only, it does not make sense to have a rule that says, one file should only contain one class.

• One file (module) should contain classes / functions that belong together, i.e. provide similar functionality or depend on each other.

• Earlier, we wrote our code in a file (module) without using classes / functions.

• In order to use classes, we will need to add a few things to our module:

o Start our code with the word class, writing the word class with any word to define the class.

o Create a main function, add your logic into the function (for example; print), we are using the words def & self, which will be explained later.

o Create an entry point to our program which will call our main function.

```
def main():
        print("hello")


if __name__ == "__main__":
    main()
```

Functions
https://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/functions.html

• Functions are a convenient way to divide your code into useful blocks.

• A function usually consists of a sequence of statements to perform an action, and possibly an output value (called the return value) of some kind.
• Functions in python are defined using the keyword "def", followed with the function's name.

• Main advantages:

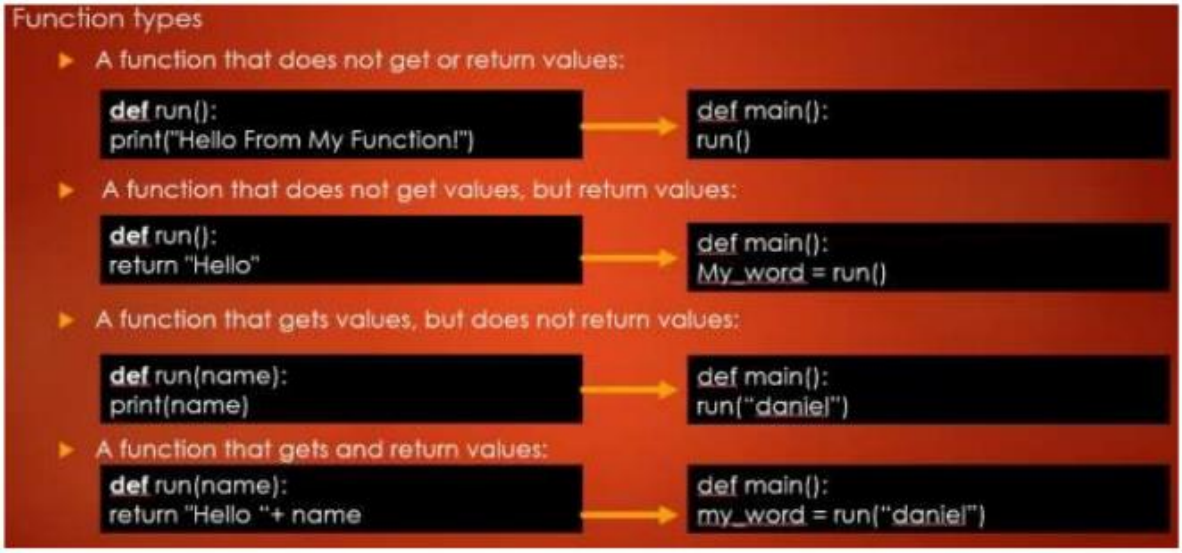
o Code reusability. O

Code optimization. o

Readable code

```
def main():
    print("hello")


if __name__ == "__main__":
  main()
```



Return, pass & yield
Return can be used to return a value as well as stopping function execution.

```
def r():
if 1 > 0:
    print(1)
    return
if 1 > 0: # will never run!
    print(2)
```

Pass can be used to continue function execution. When it is executed, nothing happens. It is useful as a placeholder when a statement is required syntactically.

```
def still_developing():
    pass
```

yield can be used to suspend function execution and sends a value back to the caller, but enable the function to resume where it is left.

```
import time
def simple_fun():
    yield 1
    time.sleep(1)
    yield 2

for value in simple_fun():
    print(value)
```

Variables

Python Programming language defines 2 kinds of variables:

a. Global variables

b. Local Variables

• Global variables (known also as Instance variables) are variables that are declare inside a class but outside any function, constructor or block or property.
• The idea is a variable that can be accessible anywhere in the class.
• In the following example name is an instance variable of Person class.

```
name = "john"

def main():
        print(name)

def main2():
        print(name)
```

• Local variables are declared in functions, constructor or blocks.
• Local variables are initialized when function or constructor block start and will be destroyed once its end.
• The variable will only be accessible from within the function.
• At the example below x is a local variable, and therefore printName can't use it.

```
def getName():
            x = 1

def printName():
            prnt(x)
```

Input

• There are hardly any programs without any input.

• Input can come in various ways, for example from a database, server and files.

• Python provides the function input(). input has an optional parameter, which is the prompt string.

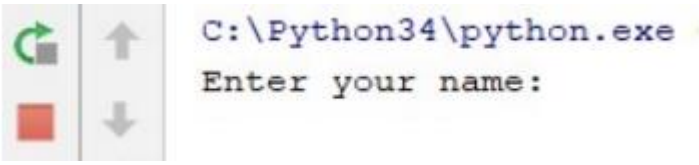
• If the input function is called, the program flow will be stopped until the user has given an input and has ended the input with the return (enter) key.

• The input of the user will be interpreted, for example if the user type in an integer value, the input function returns this integer value.

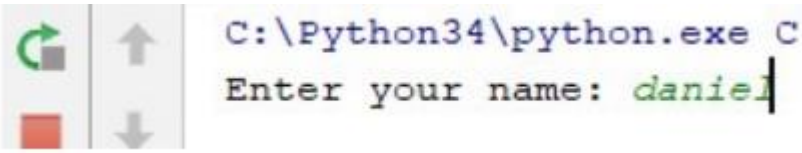
• Let's see:

```
name = input("Enter your name: ")
print("Your name is:", name)
```

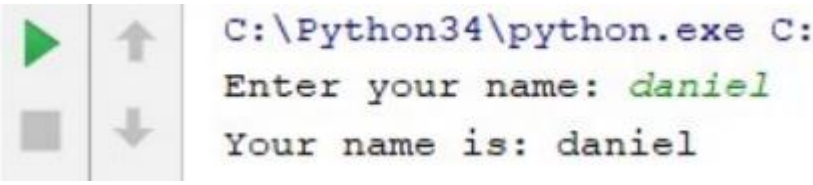
• When you will run your program, you will see the following inside the console:



• Press with the mouse underneath, and type your name:



• Once you are done press return (enter), and you will see the output:



Data structure:

https://docs.python.org/3/tutorial/datastructures.html

List:

• Lists in Python are used to store collection of heterogeneous items.

• These are mutable, which means that you can change their content without changing their identity.

• You can recognize lists by their square brackets [and] that hold elements, separated by a comma:

```
my_list = [5,"a",True]
```

• List shares the same functions as array!

List usage:

• Adding an element

```
a = [5,"a",True]
a.append(111)
```

• Removing an element

```
a = [5,"a",True]
a.pop(0)
```

• Modify existing value in a specific index

```
a = [5,"a",True]
a[0] = 5
```

• Get a value from a specific index

```
a = [5,"a",True]
print(a[0])
```

• Adding an element to a specific index

```
a = [5,"a",True]
a.insert(1,7)
```

• Get the array size (number of elements) using **len**

```
a = [5,"a",True]
print(len(a))
```

• Iterating the array and getting all elements without index

```
a = [5,"a",True]
for temp in a:
    print(temp)
```

• Without index

```
a = [5,"a",True]
for i in range(len(a)):
    print(a[i])
```

Tuple:

• Tuple is a data structure very similar to the **list** data structure.

• The main difference being that tuple manipulation are faster than list because tuples are immutable, which means once defined you cannot delete, add or edit any values inside it.

• The simple usage can be, when we already know what data is stored, and we don't want it to change (for example: seasons)

• Another usage is in situations where you want to pass the data to someone else but you do not want them to manipulate data in your collection.

• Tuple can be written with/without brackets:

```
x_tuple = 1, 2, 3, 4, 5
y_tuple = ('a','b','c','d')
```

Dictionary:

• A dictionary is a sequence of items where each item is a pair made of a key and a value.

• Dictionaries are not sorted, so you can access to the list of keys or values independently.

• Dictionaries are surrounded by curly brackets

```
my_dictionary = {'A': 1, 'B': 2, 'C': 3, 'D': 4}
```

• Dictionaries come with many build in options, such as changing a specific value

```
my_dictionary['A'] = 5
```

• Getting all dictionary keys/values

```
Print(my_dictionary.keys()) → Print(my_dictionary.values())
```

• Deleting a pair

```
del(my_dictionary['A'])
```

• A full list of data structures functions can be found here:
https://docs.python.org/3.3/tutorial/datastructures.html