MODULE-2

PPT-5

Scoping

- A variable is only available from inside the region it is created. This is called scope.
- Local Scope- A variable created inside a function belongs to the local scope of that function, and can only be used inside that function.

```
def myfunc():
  x = 300
  print(x)
myfunc()
```

Function Inside Function

- The local variable can be accessed from a function within the function:

```
def myfunc():
  x = 300
  def myinnerfunc():
    print(x)
  myinnerfunc()

myfunc()
```

Global Scope

- A variable created in the main body of the Python code is a global variable and belongs to the global scope.
- Global variables are available from within any scope, global and local.

- x = 300
- def myfunc():
-   print(x)
- myfunc()
- print(x)

Naming Variables

- If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function):

```
x = 300
def myfunc():
  x = 200
  print(x)
myfunc()
print(x)
```

Global Keyword

- If you need to create a global variable, but are stuck in the local scope, you can use the global keyword. The global keyword makes the variable global.
- def myfunc():
-   global x
-   x = 300
- myfunc()
- print(x)

Modules

- A file containing a set of functions you want to include in your application.
- Create a Module-To create a module just save the code you want in a file with the file extension .py:

- Save this code in a file named mymodule.py

- def greeting(name):
-   print("Hello, " + name)

Use a Module

- Now we can use the module we just created, by using the import statement: Import the module named mymodule, and call the greeting function:

- import mymodule
- mymodule.greeting("Jonathan")

Variables in Module

- The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):
- Save this code in the file mymodule.py
- person1 = {
-   "name": "John",
-   "age": 36,
-   "country": "Norway"
- }

Import the module named mymodule, and access the person1 dictionary:

- import mymodule
- a = mymodule.person1["age"]
- print(a)

- Re-naming a Module- Create an alias for mymodule called mx:
- import mymodule as mx
- a = mx.person1["age"]
- print(a)

- Built-in Modules-Import and use the platform module:

- import platform
- x = platform.system()
- print(x)

Import From Module

- You can choose to import only parts from a module, by using the from keyword.
- The module named mymodule has one function and one dictionary:
- def greeting(name):
-   print("Hello, " + name)
- person1 = {
-   "name": "John",
-   "age": 36,
-   "country": "Norway"
- }

Import only the person1 dictionary from the module:

- from mymodule import person1
- print (person1["age"])

Exception Handling

- The try block lets you test a block of code for errors.
- The except block lets you handle the error.
- The finally block lets you execute code, regardless of the result of the try- and except blocks.
- When an error occurs, or exception as we call it, Python will normally stop and generate an error message.
- These exceptions can be handled using the try statement:
- The try block will generate an exception, because x is not defined:

```
try:
  print(x)
except:
  print("An exception occurred")
```

Many Exceptions

- You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error.
- Print one message if the try block raises a NameError and another for other errors:

```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

Else

- You can use the else keyword to define a block of code to be executed if no errors were raised:
- In this example, the try block does not generate any error:

```
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```

Finally

- The finally block, if specified, will be executed regardless if the try block raises an error or not.

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

Raise an exception

- As a Python developer you can choose to throw an exception if a condition occurs.
- To throw (or raise) an exception, use the raise keyword.
- Raise an error and stop the program if x is lower than 0:

```
x = -1
if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

- The raise keyword is used to raise an exception. You can define what kind of error to raise, and the text to print to the user.
- Raise a TypeError if x is not an integer:

```
x = "hello"
if not type(x) is int:
  raise TypeError("Only integers are allowed")
```