DA1: DESIGNING APPLICATIONS IN PYTHON LECTURE #2

Aidos Sarsembayev

CONTENT

- Tuples, Dictionaries
- Errors handling
- Conditionals
- Loops and User Input
- File Handling
- Modules, libraries, packages
- Date and time

TUPLES

- Tuples are like lists, except that they are not mutable (changeable):

```
tuple = ('Winterfell', 'The Wall')

print('the tuple: ',tuple)

>> the tuple:  ('Winterfell', 'The Wall')
```

- you can access an item in the tuple by simple indexing

```
print(tuple[0])

>> Winterfell
```

TUPLES

- You can find all the methods applied to tuples by executing *dir(tuple)* in terminal:

```
>>> dir(tuple)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmul__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', 'count', 'index']
```

DICTIONARIES

- The dictionaries are similar to the HashMaps in other programming languages. The dictionaries are the key-value pairs

```
dictionary = {"Name":"Jon", "Surname":'Snow', "Cities":("Winterfell", "The Wall")}

print(dictionary)

>>  {'Name': 'Jon', 'Surname': 'Snow', 'Cities': ('Winterfell', 'The Wall')}
```

- you can access the item(s) of a dictionary by calling the key value:

```
print(dictionary["Name"])

>> Jon

print(dictionary["Cities"])

>> ('Winterfell', 'The Wall')

print(dictionary["Cities"][1])

>> The Wall
```

DICTIONARIES

```
print(dictionary.__contains__('Cities'))
>> True
print(dictionary.__contains__('The Wall'))
>> False
print(dictionary["Cities"].__contains__('The Wall'))
>> True
print(dictionary.__len__())
>> 3
```

DICTIONARIES

```
>>> dir(dict)
['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__setitem__', '__sizeof__', '__str__',
'__subclasshook__', 'clear', 'copy', 'fromkeys',  'get', 'items', 'keys', 'pop', 'popitem', 'setdefault',
'update', 'values']
```

THE LISTS COMMANDS

```
>>> dir(list)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__',
'__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__',
'__rmul__', '__setattr__', '__setitem__',  '__sizeof__', '__str__', '__subclasshook__', 'append',
'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

ERRORS HANDLING (SYNTAXERROR(1))

- There are two types of errors in Python: **Syntax Errors** and **Exceptions**

```
print(1)
int(234)
#int 234

>>  File ".\errors_handling.py", line 5    # this tells you were to search the error for
    int 234                                 # this shows you the exact place of the error
            ^                       # the arrow points you at the token with an error or on the end of line
SyntaxError: invalid syntax             # this shows you the type of the error - SyntaxError
```

ERRORS HANDLING (SYNTAXERROR(2))

- print 234

```
>>>
  File ".\errors_handling.py", line 13
    print 234
                ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(234)
```

ERRORS HANDLING (SYNTAXERROR(3))

- a = [1,2,3)

```
File ".\errors_handling.py", line 20
    a = [1,2,3)
                ^
SyntaxError: invalid syntax
```

ERRORS HANDLING (SYNTAXERROR(4))

- a = [1,2,3)

File ".\errors_handling.py", line 20

```
    a = [1,2,3)
                ^
```

SyntaxError: invalid syntax

ERRORS HANDLING (EXCEPTIONS(1))

```
a = 1
b = "2"
print(int(2.5)
print(a+b)
>>
 File ".\errors_handling.py", line 34
    print(a+b)
          ^
```

SyntaxError: invalid syntax

ERRORS HANDLING (EXCEPTIONS(1))

```
a = 1
b = "2"
print(int(2.5)
print(a+b)
>>
 File ".\errors_handling.py", line 34
    print(a+b)
          ^
```

SyntaxError: invalid syntax

Why do we got SyntaxError here?

Because Python always searches for the SyntaxErrors first

Notice the position of the arrow pointing at the error

ERRORS HANDLING (EXCEPTIONS(2))

```
a = 1
b = "2"
print(int(2.5))
print(a+b)
```

Now when we fixed the error we get:

```
>> Traceback (most recent call last):
  File ".\errors_handling.py", line 34, in <module>
    print(a+b)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Traceback is an Exception error

ERRORS HANDLING (EXCEPTIONS(3))

```
#print(a/0)
>>>
Traceback (most recent call last):
  File ".\errors_handling.py", line 55, in <module>
    print(a/0)
ZeroDivisionError: division by zero
```

ERRORS HANDLING (EXCEPTIONS(3))

```
def divide(a,b):
    return a/b


print(divide(4,2))
print(divide(4,0))
```

ERRORS HANDLING (EXCEPTIONS(3))

We can handle the errors by using try-except block:

```
def divide2(a,b):
    try:
        return a/b
    except ZeroDivisionError:
        return "zero division is not allowed"


print(divide2(4,0))
```

CONDITIONALS

```
if some_condition:
        print('Some condition holds')
elif other_condition:
        print('Other condition holds')
else:
        print('Neither condition holds')
```

- else is optional
- An Aside Python has no switch statement, opting for if/elif/else chains

LOOPS AND USER INPUT (1)

- There are two types of loops in Python: ***for*** and ***while***

```
for item in iterable:
        process(item)
```

Loop explicitly over data (for …here… in)

Strings, lists, etc. (iterable)

No loop counter!

LOOPS AND USER INPUT (2)

```
emails = ["me@mail.com","he@hotmail.com","they@gmail.com"]


for email in emails:
    print(email)
```

>>

me@mail.com

he@hotmail.com

they@gmail.com

LOOPS AND USER INPUT (3)

You can combine loops w/ the conditionals:

```
for email in emails:
    if 'gmail' in email:
        print('yes, there is a gmail acc ',email)
>>
yes, there is a gmail acc  they@gmail.com
```

LOOPS AND USER INPUT (4)

Sometimes you want to use a for loop for more than one list:

```
name = ['me', 'he', 'she', 'we','them']
domain = ['mail.com','hotmail.com','gmail.com', 'mail.ru']
for i,j in zip(name, domain):       # https://docs.python.org/3/library/functions.html#zip
    print(i,'@',j)
>>
me @ mail.com
he @ hotmail.com
she @ gmail.com
we @ mail.ru
```

LOOPS AND USER INPUT (4)

Sometimes you want to use a for loop for more than one list:

```
name = ['me', 'he', 'she', 'we','them']
domain = ['mail.com','hotmail.com','gmail.com', 'mail.ru']
for i,j in zip(name, domain):       # https://docs.python.org/3/library/functions.html#zip
    print(i,'@',j)
>>
me @ mail.com
he @ hotmail.com
she @ gmail.com
we @ mail.ru
```

zip() function iterates until one of the lists exhaust

LOOPS AND USER INPUT (4)

You can use more than two lists

```
name = ['me', 'he', 'she', 'we','them']
domain = ['mail.com','hotmail.com','gmail.com', 'mail.ru']
nzzz = ['Aidos','Olzhas','Asel']
surzzz = ['A','B','C','D','E','F','G']
for i,j,k,l in zip(name, domain, nzzz, surzzz):
    print(i,'@',j,k,l)
>>
```

me @ mail.com Aidos A

he @ hotmail.com Olzhas B

she @ gmail.com Asel C

LOOPS AND USER INPUT (5)

```
names = {"Males“ : ["Jon", "Rob", "Stanis", "Jorah", "Drogo"],
"Females“ : ["Dayneris", "Sansa", "Ariya"]}
name = input('What is your character? ')


for key in names.keys():
    for item in names[key]:
        if item == name:
            print(item, ' is a ', key[:-1])
```

LOOPS AND USER INPUT (5)

```
names = {"Males“ : ["Jon", "Rob", "Stanis", "Jorah", "Drogo"], "Females“ : ["Dayneris", "Sansa", "Ariya"]}
name = input('What is your character? ')


for key in names.keys():
    for item in names[key]:
        if item == name:
            print(item, ' is a ', key[:-1])
```

Sansa

>>

Sansa is a Female

LOOPS AND USER INPUT (6) WHILE LOOP

```
password = ""
while password != "123p":
    password = input('enter a password: ')
    if password == "123p":
        print('you are logged in')
    else:
        print('try again')
```

FILE HANDLING (1)

```
file = open('Readme.txt', 'r')
print('ready')
content = file.read()
print(content)
```

FILE HANDLING (2) READING

```
file = open('Readme.txt', 'r')
lines = file.readlines()          # returns a list with lines separated by the commas
print('lines ',lines)
print('lines ',lines[10])
```

FILE HANDLING (3) READING

If you would like to get rid of the newline charachter, you can apply rstrip method:

```
file = open('Readme.txt', 'r')

lines = file.readlines()

# returns a list with lines separated by the commas
https://docs.python.org/3/library/stdtypes.html?highlight=rstrip#str.rstrip

lines = [i.rstrip("\n") for i in lines]

print('lines ',lines)

file.close()
```

FILE HANDLING (3) READING

Always close the stream by using close() method:

file.close()

Unless you are using ***with*** statement:

WITH STATEMENT

- ***with*** statement allows you to write clear code when handling the files
- It allows you not to close() the file in the end of operations
- It is similar to for loop in some terms

```
with open('new_readme.txt','r') as file:
    content = file.read()
    print(content)
```

FILE HANDLING (4) WRITING

```
line1 = 'Line1\n'
line2 = 'Line2\n'
line3 = 'Line3\n'

write_file = open('new_readme.txt', 'w')
# if we put the parameter 'w' to the open method we will overwrite the existing lines

write_file.write(line1)
write_file.write(line2)
write_file.write(line3)

write_file.close()
```

FILE HANDLING (5) WRITING

```
line1 = 'Line1'
line2 = 'Line2'
line3 = 'Line3'

write_file = open('new_readme.txt', 'a')
# if we put the parameter 'a' (append) to the open method we will append to the existing lines

write_file.write(line1)
write_file.write(line2)
write_file.write(line3)

write_file.close()
```

MODULES, LIBRARIES, PACKAGES

- Modules in python are simply the files
- whereas libraries are the folders containing modules and files
- # C:\Users\Айдос\AppData\Local\Programs\Python\Python36
- Packages are the third-party libraries
- In order to install a package you need to execute 'pip install package name' in terminal

OS MODULE EXAMPLE

```
import os

files = os.listdir()

print(files)

>>

['.ipynb_checkpoints', '1.py', 'dates_&_times.py', 'dictionaries_and_tuples.py', 'errors_handling.py',
'ext_libs.py', 'first.py', 'functions.py', 'guidelines.txt', 'loops.py', 'new_readme.txt',
'Readme.txt','txt_files_reading.py', 'txt_file_writing.py', 'withStatement.py']

---------------------------------------------------------------------------------------------------------------

path = os.getcwd()

print(path)

>>

D:\WORK\TEACHING\Des.App.Py\1_
```

DATE AND TIME (1)

- The datetime is a built in module. It grabs time from your OS

```
import datetime as dt
print(dt.datetime.now())
print(type(dt.datetime.now()))
>>
2018-09-10 00:24:19.340851
<class 'datetime.datetime'>
```

DATE AND TIME (2)

- If you would like to substitute current time from yesterday's:

```
yesterday = dt.datetime(2018, 9, 8, 23, 7, 18)
now = dt.datetime.now()
delta = now - yesterday
print(delta)
print(delta.total_seconds())
>>
1 day, 1:17:01.340851
91021.340851
```

DATE AND TIME (3)

- Now let's say we want to save a file with the file name consisting of the date
- For that we use ***strftime*** function. please refer to http://strftime.org/ in order to see the format

```
str_file_name = now.strftime("%Y-%m-%d-%H-%M-%S")
with open(str_file_name + ".txt",'w') as file:
    file.write("this is an example")
    print('file is saved')
```