2020

Free Python Course

O. Elijah
onlinepythonclass@gmail.com

1 PROJECT – 2

In this chapter, we shall take a look at converting the python codes in your .py file into an executable file and how to make your codes robust, i.e free of bugs.

1.1 Review of Chapter 8

In chapter 8 we looked at the software development life cycle. In addition, we looked at system development cycle, approach to solving a problem and a mini project.

1.2 Making your codes robust

So far, we have learnt how to write codes and how to develop projects using python codes. However, we need to bear in mind that there are different kinds of errors in programming. They include

1. **Syntax errors:** errors due to the fact that the syntax of the language is not respected.
2. **Semantic errors**: errors due to an improper use of program statements.
3. **Logical errors:** errors due to the fact that the specification is not respected. This is the most difficult error or bug to fix in your code.

The above types of errors can be detected in the following ways

1. **Compile time errors:** syntax errors and static semantic errors indicated by the compiler.
2. **Runtime errors:** dynamic semantic errors, and logical errors, that cannot be detected by the compiler (debugging).

Let us look at examples of the different types of errors in python

Most common is the syntax error which results to compile time errors.

```
print "hello"

SyntaxError: Missing parentheses in call to 'print'. Did you mean
print("hello")?
```

Other examples of run time errors include

Index error is thrown when trying to access an item at an invalid index.

```
L1=[1,2,3]
L1[3]

Traceback (most recent call last):
File "<pyshell#18>", line 1, in <module>
L1[3]
IndexError: list index out of range
```

ModuleNotFoundError is thrown when a module could not be found.

```
import notamodule

Traceback (most recent call last):
File "<pyshell#10>", line 1, in <module>
import notamodule
ModuleNotFoundError: No module named 'notamodule'
```

KeyError is thrown when a key is not found.

```
D1={'1':"aa", '2':"bb", '3':"cc"}
D1['4']

Traceback (most recent call last):
File "<pyshell#15>", line 1, in <module>
D1['4']
KeyError: '4'
```

TypeError is thrown when an operation or function is applied to an object of an inappropriate type.

```
'2'+2

Traceback (most recent call last):
File "<pyshell#23>", line 1, in <module>
'2'+2
TypeError: must be str, not int
```

NameError is thrown when an object could not be found.

```
age

Traceback (most recent call last):
File "<pyshell#6>", line 1, in <module>
age
NameError: name 'age' is not defined
```

KeyboardInterrupt is thrown when the user hits the interrupt key (normally Control-C) during the execution of the program.

```
name=input('enter your name')

enter your name^c
Traceback (most recent call last):
File "<pyshell#9>", line 1, in <module>
name=input('enter your name')
KeyboardInterrupt
```

To make your codes robust, i.e free of bugs you can use what the call **exception handling.** There are several built-in exception handlings in python.

Python uses **try** and **except** keywords to handle exceptions. Both keywords are followed by indented blocks.

The syntax is

```
try :
    #statements in try block
except :
    #executed when error in try block
```

The **try**: block contains one or more statements which are likely to encounter an exception. If the statements in this block are executed without an exception, the subsequent **except**: block is skipped.

If the exception does occur, the program flow is transferred to the **except**: block. The statements in the **except**: block are meant to handle the cause of the exception appropriately. For example, returning an appropriate error message.

Example

```
try:
    a=5
    b='0'
    print (a+b)
except TypeError:
    print('Unsupported operation')
print ("Out of try except blocks")
```

Result:

```
Unsupported operation

Out of try except blocks
```

A single **try** block may have multiple **except** blocks.

Example

```
try:
    a=5
    b=0
    print (a/b)
except TypeError:
    print('Unsupported operation')
except ZeroDivisionError:
    print ('Division by zero not allowed')
print ('Out of try except blocks')
```

Result

```
Division by zero not allowed

Out of try except blocks
```

Raise an Exception

Python also provides the **raise** keyword to be used in the context of exception handling. It causes an exception to be generated explicitly. Built-in errors are raised implicitly. However, a built-in or custom exception can be forced during execution.

The following code accepts a number from the user. The try block raises a ValueError exception if the number is outside the allowed range.

```
try:
    x=int(input('Enter a number upto 100: '))
    if x > 100:
        raise ValueError(x)
except ValueError:
    print(x, "is out of allowed range")
else:
    print(x, "is within the allowed range")
```

Result

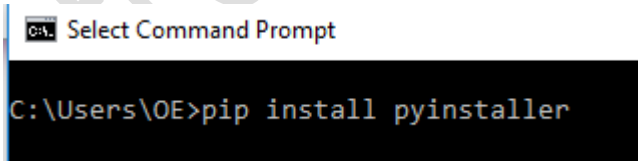
```
Enter a number upto 100: 200
200 is out of allowed range
Enter a number upto 100: 50
50 is within the allowed range
```

1.3 Converting your python codes to executable file

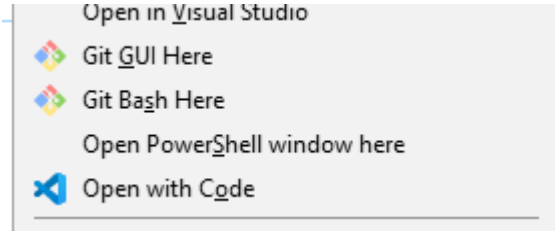
Executable files are used to share your developed application with people or your clients. This may include people who do not have python interpreter on their PC or devices. Hence, the need to install the executable file.

Here is simple guide on how to convert your python codes into an executable file.

First you need to install the **Pyinstaller** by opening the command prompt on your system



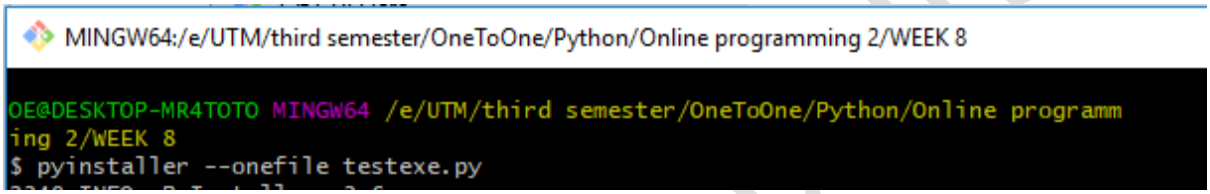
Second, navigate to the directory or folder where your code is located and press down the shift key on your keyboard and right click and open the powershell or cmd window or Git Bash



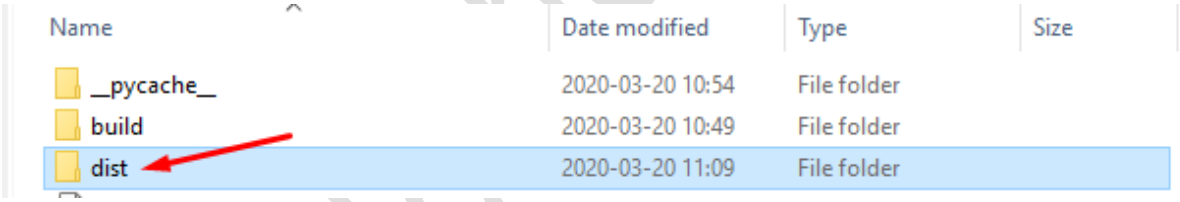
Visit this link for more details https://pythonprogramming.altervista.org/create-an-exe-with-pyinstaller/

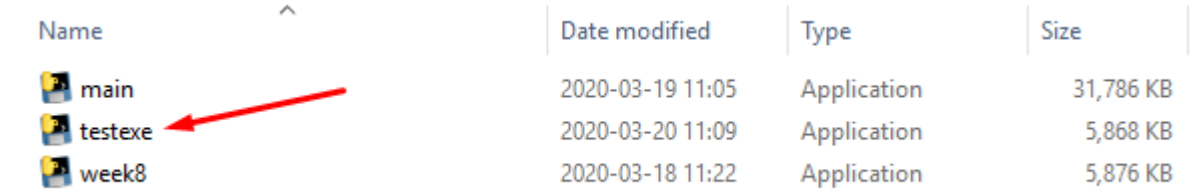
Ones the cmd window opens, you can run the following command

pyinstaller –onefile testexe.py as show in the following figure. Note the testexe.py is the name of the file you want to convert into executable.



Ones it has successfully completed, you can open the folder where the python file is and open the **dist** folder, and open your executable file. Double click on the executable file to run.





Note, if you have multiple files with dependency, you need to create a hook file to link them together otherwise the executable file will not run. We shall see how to implement that in the intermediate course.

1.4 Application

Let us create an app that consist of login system.

In this apps, the password should be hidden to the users when they login. See the codes as follows.

Note the getpass function helps to hide the password while the user enters the password. But on the console, you can still see the password while you type.

Also, the username in this case is **admin** and password is **password**

```
from getpass import getpass

def login():

    username = input("Please enter your username:  ")

    password = getpass("Please enter your passowrd: ")


    return [username, password]

try:

    login_details = login()

    if login_details[0] != "admin" and login_details[1] != "password":

        raise ValueError()

except ValueError:

    print("Wrong User name or password")

else:

    print("Login successful")

input("Press enter to exit")
```

Result

```
Please enter your username: admin

Warning: QtConsole does not support password mode, the text you type will be visible.
Please enter your passowrd: password
Login successful

Press enter to exit
```

1.5 Reference

[1] https://www.tutorialsteacher.com/python/error-types-in-python

[2] https://www.youtube.com/watch?v=hk3ubc1-ZGg

1.6 Exercise

1. Modify the codes in the Application section so that you can detect whether the password or username is incorrect
2. Create an executable file for the code
3. Submit the .py and executable file to LMS

Due Date is on the 28th March