Computer Programming I

Binnur Kurt, PhD

BAU Bahçeşehir University

binnur.kurt@rc.bau.edu.tr

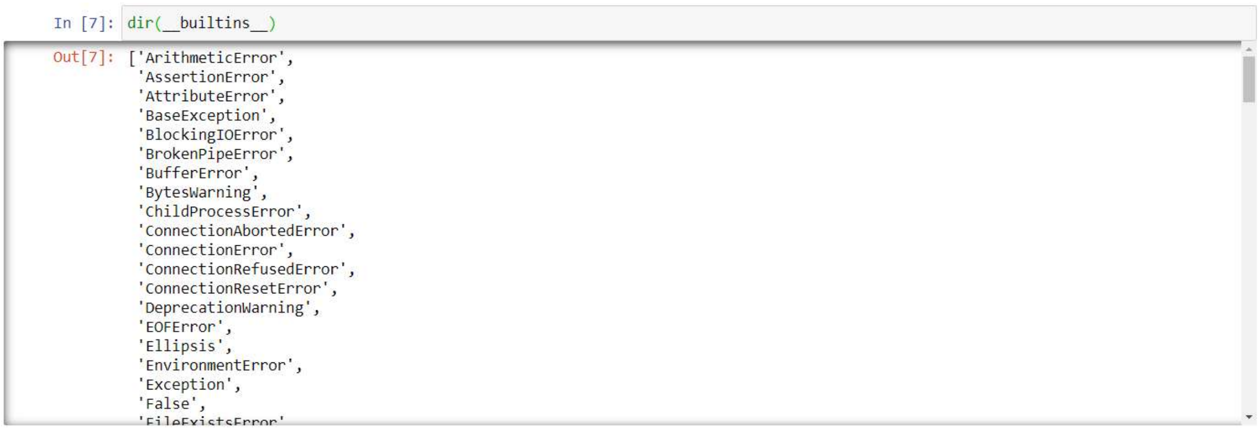
MODULE 12

MODULES AND PACKAGES

Exploring Modules

- A module is just a Python source file.
- The module can contain variables, classes, functions, and any other element available in your Python scripts.
- You can get a better understanding of modules by using the **`dir`** function.
- Pass the name of some Python element, such as a module, and **`dir`** will tell you all of the attributes of that element.
- For example, to see the attributes of **`__builtins__`**, which contain built-in functions, classes, and variables, use **`dir(__builtins__)`**

Exploring Modules



```
In [7]: dir(__builtins__)

Out[7]: ['ArithmeticError',
         'AssertionError',
         'AttributeError',
         'BaseException',
         'BlockingIOError',
         'BrokenPipeError',
         'BufferError',
         'BytesWarning',
         'ChildProcessError',
         'ConnectionAbortedError',
         'ConnectionError',
         'ConnectionRefusedError',
         'ConnectionResetError',
         'DeprecationWarning',
         'EOFError',
         'Ellipsis',
         'EnvironmentError',
         'Exception',
         'False',
         'FileExistsError',
```

Importing Modules

> Before using a module, you need to import it.

> The standard syntax for importing

```
import module
```

> You can use this syntax with modules that come with Python or with modules you create.

> You can also use the following alternative syntax:

```
from module import item
```

> The alternative syntax enables you to specifically import just a class or function if that is all you need.

Importing Modules

> If a module has changed, you can reload the new definition of the module using the **`imp.reload`** function:

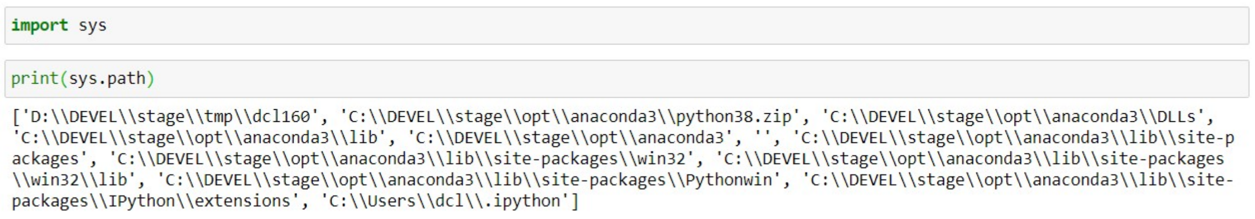
```
import module
import imp
imp.reload(module)
```

Finding Modules

- When you place an import statement in your scripts, the Python interpreter has to be able to find the module.
- The key point is that the Python interpreter only looks in a certain number of directories for your module.
- The Python interpreter looks in the directories that are part of the module search path.
- These directories are listed in the **`sys.path`** variable from the **`sys`** module.

Finding Modules

- To list where the Python interpreter looks for modules, print out the value of the **`sys.path`** variable in the Python interpreter.



```
import sys
```

```
print(sys.path)
```

```
['D:\\DEVEL\\stage\\tmp\\dcl160', 'C:\\DEVEL\\stage\\opt\\anaconda3\\python38.zip', 'C:\\DEVEL\\stage\\opt\\anaconda3\\DLLs',
'C:\\DEVEL\\stage\\opt\\anaconda3\\lib', 'C:\\DEVEL\\stage\\opt\\anaconda3', '', 'C:\\DEVEL\\stage\\opt\\anaconda3\\lib\\site-p
ackages', 'C:\\DEVEL\\stage\\opt\\anaconda3\\lib\\site-packages\\win32', 'C:\\DEVEL\\stage\\opt\\anaconda3\\lib\\site-packages
\\win32\\lib', 'C:\\DEVEL\\stage\\opt\\anaconda3\\lib\\site-packages\\Pythonwin', 'C:\\DEVEL\\stage\\opt\\anaconda3\\lib\\site-
packages\\IPython\\extensions', 'C:\\Users\\dcl\\.ipython']
```

Creating Modules and Packages

> A module is merely a Python source file.

> Any time you have created a Python file, you have already been creating modules without even knowing it.

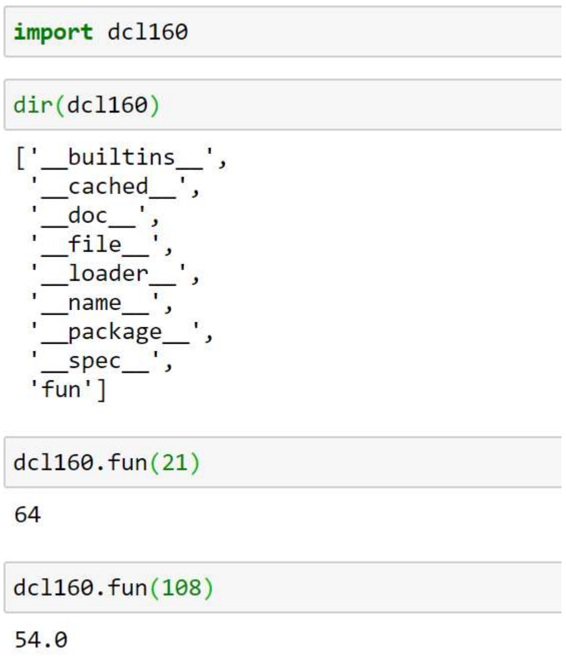
dcl160.py

```
def fun(x):
   return x/2 if x%2 == 0 else   3 * x + 1
```

Creating Modules and Packages

dcl160.py

```
def fun(x):
   return x/2 if x%2 == 0 else 3 * x + 1
```



```
import dcl160
```

```
dir(dcl160)
```

```
['__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'fun']
```

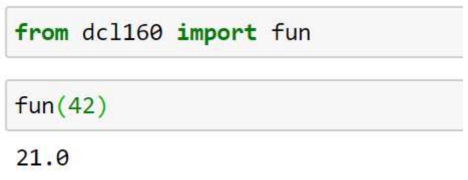
```
dcl160.fun(21)
```

```
64
```

```
dcl160.fun(108)
```

```
54.0
```

Creating Modules and Packages

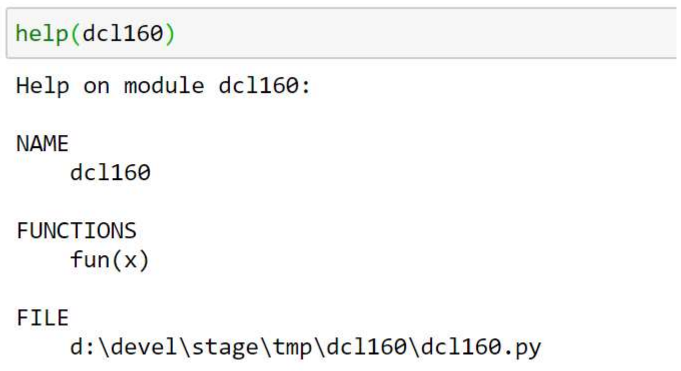


```
from dcl160 import fun
```

```
fun(42)
```

```
21.0
```

Viewing Module Documentation

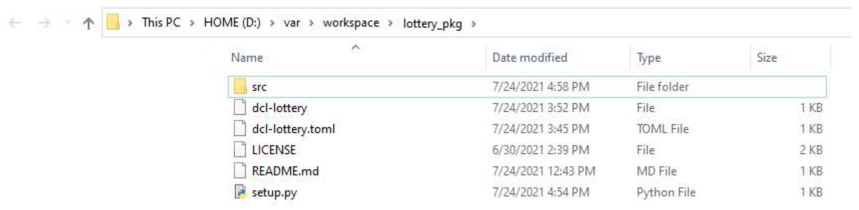


```
help(dcl160)
```

```
Help on module dcl160:

NAME
    dcl160

FUNCTIONS
    fun(x)

FILE
    d:\devel\stage\tmp\dcl160\dcl160.py
```

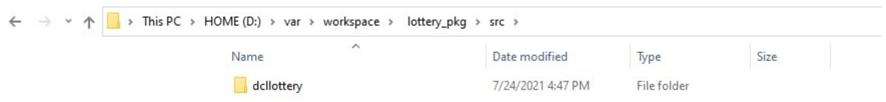
How to create a Python package

> The main difference between a module and a package is that a package is a collection of modules AND it has an __init__.py file.



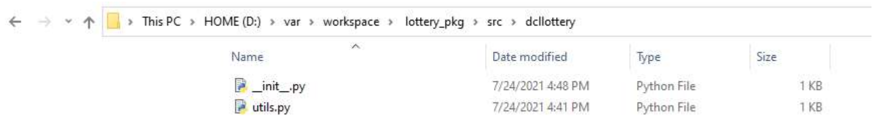
This PC › HOME (D:) › var › workspace › lottery_pkg ›

Name	Date modified	Type	Size
src	7/24/2021 4:58 PM	File folder	
dcl-lottery	7/24/2021 3:52 PM	File	1 KB
dcl-lottery.toml	7/24/2021 3:45 PM	TOML File	1 KB
LICENSE	6/30/2021 2:39 PM	File	2 KB
README.md	7/24/2021 12:43 PM	MD File	1 KB
setup.py	7/24/2021 4:54 PM	Python File	1 KB



This PC › HOME (D:) › var › workspace › lottery_pkg › src ›

Name	Date modified	Type	Size
dcllottery	7/24/2021 4:47 PM	File folder	



This PC › HOME (D:) › var › workspace › lottery_pkg › src › dcllottery

Name	Date modified	Type	Size
__init__.py	7/24/2021 4:48 PM	Python File	1 KB
utils.py	7/24/2021 4:41 PM	Python File	1 KB

How to build a PIP package

> Create a **setup.py** script

```
import setuptools

with open("README.md", "r") as fh:
   long_description = fh.read()

setuptools.setup(
    name='dcllottery',
    version='0.1',
    scripts=['dcl-lottery'] ,
    author="Binnur Kurt",
    author_email="info@deepcloudlabs.com",
    description="Lottery utility package",
    long_description=long_description,
    long_description_content_type="text/markdown",
    url="https://github.com/deepcloudlabs/lottery",
    packages=setuptools.find_packages(where="src"),
    classifiers=[
       "Programming Language :: Python :: 3",
       "License :: OSI Approved :: MIT License",
       "Operating System :: OS Independent",
    ],
   package_dir={"": "src"},
   python_requires=">=3.6",
 )
```

How to build a PIP package

> Build your package

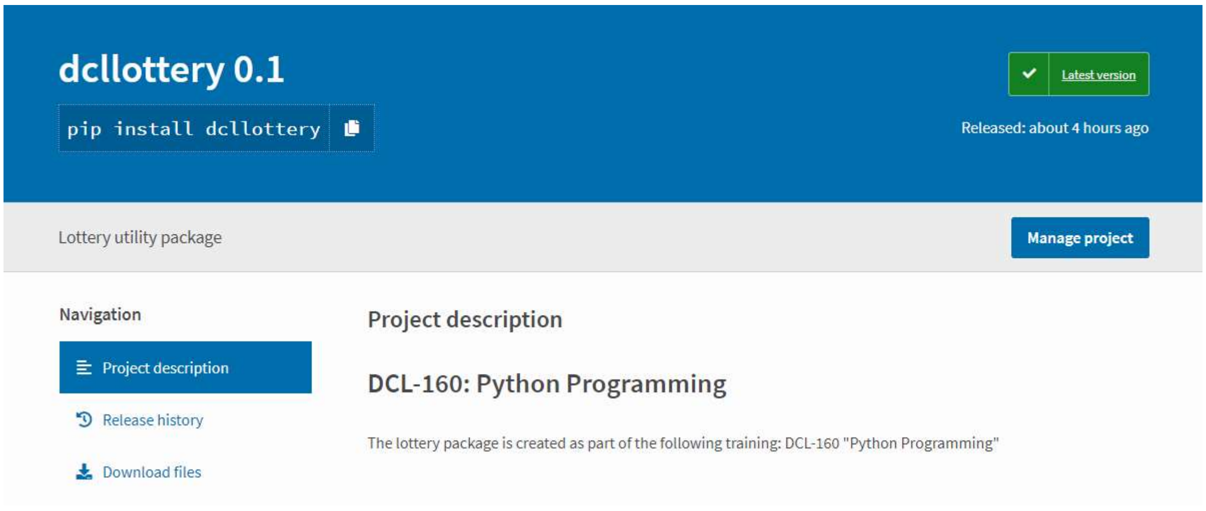
```
$ python setup.py bdist_wheel
running bdist_wheel
running build
running build_py
creating build
creating build\lib
creating build\lib\dcllottery
copying src\dcllottery\utils.py -> build\lib\dcllottery
copying src\dcllottery\__init__.py -> build\lib\dcllottery
. . .
adding 'dcllottery-0.1.dist-info/RECORD'
removing build\bdist.win-amd64\wheel
```

Uploading the distribution archives

> Finally, it's time to upload your package to the Python Package Index

```
$ python -m twine upload  dist/*
Uploading distributions to https://upload.pypi.org/legacy/
Enter your password: *your password*
Uploading dcllottery-0.1-py3-none-any.whl
100%|█████████████████████████████████████████████| 6.06k/6.06k [00:02<00:00, 2.25kB/s]

View at:
https://pypi.org/project/dcllottery/0.1/
```

https://pypi.org/project/dcllottery



dcllottery 0.1

```
pip install dcllottery
```

Latest version

Released: about 4 hours ago

Lottery utility package

Manage project

Navigation

- Project description
- Release history
- Download files

Project description

DCL-160: Python Programming

The lottery package is created as part of the following training: DCL-160 "Python Programming"

Installing newly uploaded package

> You can use pip to install your package and verify that it works

```
$ pip install dcllottery
Collecting dcllottery
  Using cached dcllottery-0.1-py3-none-any.whl (3.0 kB)
Installing collected packages: dcllottery
Successfully installed dcllottery-0.1
```

Using newly uploaded package

```
import dcllottery.utils as dcl

numbers = dcl.get_lottery_numbers(1,60,6)
print(numbers)
```