

# PIP

- PIP is a package manager for Python packages, or modules
- Package - contains all the files you need for a module
- Modules are Python code libraries you can include in your project
  - A file containing a set of functions
- If not preinstalled → Install PIP
  - Download get-pip.py
  - In command prompt : `python get-pip.py`
  - Pip is now installed
- Check if PIP is Installed
  - Navigate your command line to the location of Python's script directory, and type the following:
    - `pip --version`

# Download a Package

- Open the command line interface and tell PIP to download the package you want.
- `pip install package_name`
- Download a package named "camelcase":
  - `pip install camelcase`
  - `import camelcase`

# Using a Package

- Once the package is installed, it is ready to use.
- Import the "camelcase" package into your project.

- `import camelcase`

```
c = camelcase.CamelCase()
```

```
txt = "hello world"
```

```
print(c.hump(txt))
```

- Hello World

CamelCase Class turns strings into CamelCase

Hump is a method that takes an inputted string and turns it into camel case format.

# Remove a Package

- Use the uninstall command to remove a package
- uninstall the package named "camelcase":
- `C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip uninstall camelcase`

# List Packages

- Use the list command to list all the packages installed on your system:
- C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>**pip list**

- Result:

Package	Version
-----	
camelcase	0.2
mysql-connector	2.1.6
pip	18.1
pymongo	3.6.1
setuptools	39.0.1

# How to import modules in Python?

- import keyword
- ton of standard modules available
- [Python standard modules](#) - files are in the Lib directory inside the location where you installed Python
- Standard modules , user-defined modules
- various ways to import modules

# Python import statement

- import math

```
print("The value of pi is", math.pi)
```

- import a module (in this example- it is math) using import statement and access the definitions (in this example- it is pi) inside it using the dot operator as described above

# Import with renaming

- import a module by renaming it
- import math as m  
print("The value of pi is", m.pi)

Here, **math** module is imported as **m**,

**So now whenever want to use math module -> use m**



# Python from...import statement

- import specific names from a module without importing the module as a whole

```
# import only pi from math module  
from math import pi  
print("The value of pi is", pi)
```

- Import multiple attributes

```
>>> from math import pi, e  
>>> pi  
3.141592653589793  
>>> e  
2.718281828459045
```

# Import all names

- import all names(definitions) from a module

```
# import all names from the standard module math
from math import *
print("The value of pi is", pi)
```

The value of pi is 3.141592653589793

# Random Number

- Python has a built-in module called random that can be used to make random numbers
- Import the random module, and display a random number between 1 and 9:

- ```
import random  
print(random.randrange(1,10))
```

randint - Return random integer in range [a, b], including both end points.

randrange - Choose a random item from range. This fixes the problem with randint() which includes the endpoint.

# Python Dates

- A date in Python is not a data type of its own, but we can import a module named datetime to work with dates as date objects.
- To create a date, we can use the datetime() class (constructor) of the datetime module.
- The datetime() class requires three parameters to create a date: year, month, day.

# Writing Modules

What is a Module?

- Consider a module to be the same as a code library.
- A file containing a set of functions you want to include in your application.

# Contd ...

## Create a Module

- To create a module just save the code you want in a file with the file extension `.py`

## Use a Module

- Now we can use the module we just created, by using the import statement
- When using a function from a module, use the syntax:  
*module\_name.function\_name*

## Variables in Module

- The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc)

# use of dir() function

- There is a built-in function to list all the function names (or variable names) in a module.
- The dir() function can be used on *all* modules, also the ones you create yourself.



# Python Lambda

- A lambda function is a small anonymous function.
  - A lambda function can take any number of arguments, but can only have one expression.
  - `lambda arguments : expression`
  - The expression is executed and the result is returned
  - Create a lambda function that takes one parameter (a) and returns it.
- ```
x = lambda a : a
```

# Contd...

- A lambda function that adds 10 to the number passed in as an argument, and print the result

```
x = lambda a: a + 10
```

```
print(x(5))
```

Ans = 15

## Contd ...

- Lambda functions can take any number of arguments
- A lambda function that multiplies argument a with argument b and print the result:

```
x = lambda a, b : a * b  
print(x(5, 6))
```

Ans = 30

# Contd ...

- A lambda function that sums argument a, b, and c and print the result:

```
x = lambda a, b, c : a + b + c  
print(x(5, 6, 2))
```

Ans = 13

# Why Use Lambda Functions?

- The power of lambda is better shown when you use them as an anonymous function inside another function.
- Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

```
def myfunc(n):  
    return lambda a : a * n
```

# Contd ...

- Use that function definition to make a function that always doubles the number you send in:

```
def myfunc(n):  
    return lambda a : a * n
```

```
mydoubler = myfunc(2)
```

```
print(mydoubler(11))
```

Ans = 22

# Contd ...

- use the same function definition to make a function that always *triples* the number you send in:
- use the same function definition to make both functions, in the same program:

```
def myfunc(n):  
    return lambda a : a * n
```

```
mydoubler = myfunc(2)  
mytripler = myfunc(3)
```

```
print(mydoubler(11))  
print(mytripler(11))
```

Ans = 22  
      33

- Use lambda functions when an anonymous function is required for a short period of time.

THANK YOU