Lecture 11- Introduction to Python Project

Guiliang Liu

The Chinese University of Hong Kong, Shenzhen

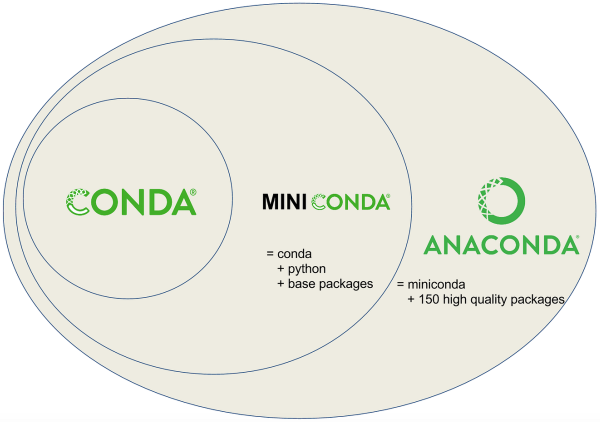
CSC-1004: Computational Laboratory Using Java
Course Page: [Click]

Outline

- Python Project Management and Basic Knowledge
- Basic Machine Learning with Python

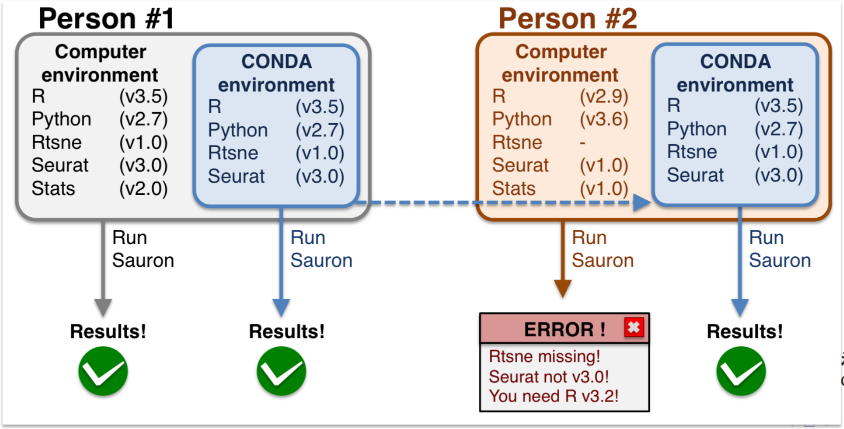
Conda

Conda is a powerful package manager and environment manager that you use with command line commands at the Anaconda Prompt for Windows, or in a terminal window for macOS or Linux.



Managing Environment with Conda

Conda allows you to create separate environments containing files, packages, and their dependencies that will not interact with other environments.



港中文大學(深圳)
Chinese University of Hong Kong, Shenzhen

Managing Environment with Conda

- **Step 1**: Create a new environment and install a package in it.

  1) We will name the environment snowflakes and install the package BioPython.

  At the Anaconda Prompt or in your terminal window, type the following:

  ```
  conda create –name snowflakes biopython
  ```

  2) Conda checks to see what additional packages ("dependencies") BioPython will need, and asks if you want to proceed:

  ```
  Proceed ([y]/n)? y
  ```

Managing Environment with Conda

- **Step 2**: To use, or "activate" the new environment, type the following:

```
conda activate snowflakes
```

- **Step 3**: To see a list of all your environments, type:

```
conda info –envs
```

A list of environments appears, similar to the following:

```
conda environments:
base /home/username/Anaconda3
snowflakes * /home/username/Anaconda3/envs/snowflakes
```

香 港 中 文 大 學 (深 圳)
The Chinese University of Hong Kong, Shenzhen

Managing Python with Conda

When you create a new environment, conda installs the same Python version you used when you downloaded and installed Anaconda. **If you want to use a different version of Python**, for example Python 3.5, simply create a new environment and specify the version of Python that you want.

- **Step 1**: Create a new environment named "snakes" that contains Python 3.9:

```
conda create –name snakes python=3.9
```

- **Step 2**: Activate the new environment:

```
conda activate snakes
```

Managing Python with Conda

- **Step 3**: Verify that the snakes environment has been added and is active:

  ```
  conda info –envs
  ```

  Conda displays the list of all environments with an asterisk (*) after the name of the active environment:

  ```
  conda environments:
  base /home/username/Anaconda3
  snakes * /home/username/anaconda3/envs/snakes
  snowflakes /home/username/Anaconda3/envs/snowflakes
  ```

- **Step 4**: Verify which version of Python is in your current environment:

  ```
  python –version
  ```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Managing packages

In this section, you check which packages you have installed, check which are available and look for a specific package and install it.

- **Step 1**: To find a package you have already installed, first activate the environment you want to search.

```
conda activate snakes
```

- **Step 2**: Check to see if a package you have not installed named "beautifulsoup4" is available from the Anaconda repository (must be connected to the Internet):

```
conda search beautifulsoup4
```

- **Step 3**: Install this package into the current environment:

```
conda install beautifulsoup4
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Managing packages

- **Step 4**:Check to see if the newly installed program is in this environment:

```
conda list
```

Python File Handling

The key function for working with files in Python is the open() function. The open() function takes two parameters; filename, and mode. There are four different methods (modes) for opening a file:

- "r" - Read - Default value. Opens a file for reading, error if the file does not exist.
- "a" - Append - Opens a file for appending, creates the file if it does not exist.
- "w" - Write - Opens a file for writing, and creates the file if it does not exist.
- "x" - Create - Creates the specified file, and returns an error if the file exists.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Python File Handling

In addition, you can specify if the file should be handled as binary or text mode.

- "t" - Text - Default value. Text mode.
- "b" - Binary - Binary mode (e.g. images).

Python Read Files

Assume we have the following file named "demofile.txt", located in the same folder:

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

Open and Read a file: To open the file, use the built-in open() function. The open() function returns a file object, which has a read() method for reading the file content:

```
f = open("demofile.txt", "r")
print(f.read())
f.close()
```

Output: The entire file.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Python Read Files

Assume we have the following file named "demofile.txt", located in the same folder:

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

Read Only Parts of the File: By default the read() method returns the whole text, but you can also specify how many characters you want to return:

```python
f = open("demofile.txt", "r")
print(f.read(5))
f.close()
```

Output:

```
Hello
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Python Read Files

Assume we have the following file named "demofile.txt", located in the same folder:

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

Read One Line of the File: You can return one line by using the readline() method:

```python
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

Output:
```
Hello! Welcome to demofile.txt
```

香 港 中 文 大 學 (深 圳)
The Chinese University of Hong Kong, Shenzhen

Python Read Files

Assume we have the following file named "demofile.txt", located in the same folder:

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

Read Multiple Lines of the File: You can return multiple lines by using the readlines() method:

```
f = open("demofile.txt", "r")
print(f.readlines())
f.close()
```

Output: a list of string.

香 港 中 文 大 學 (深 圳)
The Chinese University of Hong Kong, Shenzhen

Python Write Files

Assume we have the following file named "demofile2.txt", located in the same folder:

```
Hello! Welcome to demofile2.txt.
```

Append to a file. Open the file "demofile2.txt" and append content to the file:

```python
ff = open("demofile2.txt", "a")
ff.write("Now the file has more content!")
ff.close()
```

The updated "demofile2.txt":

```
Hello! Welcome to demofile2.txt. Now the file has more content!
```

Python Write Files

Assume we have the following file named "demofile2.txt", located in the same folder:

```
Hello! Welcome to demofile2.txt.
```

Overwrite a file. Open the file "demofile3.txt" and overwrite the content:

```
ff = open("demofile2.txt", "w")
ff.write("Woops! I have deleted the content!")
ff.close()
```

The updated "demofile2.txt":

```
Woops! I have deleted the content!
```

香 港 中 文 大 學 (深 圳)
The Chinese University of Hong Kong, Shenzhen

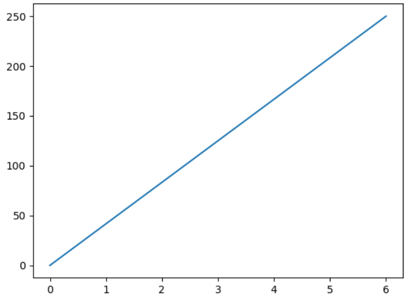
Python Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

Matplotlib Pyplot

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

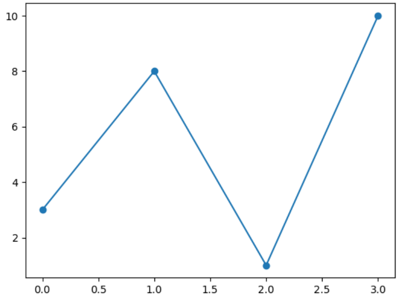
```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([0, 6])
ypoints = np.array([0, 250])
plt.plot(xpoints, ypoints)
plt.show()
```



Matplotlib Markers

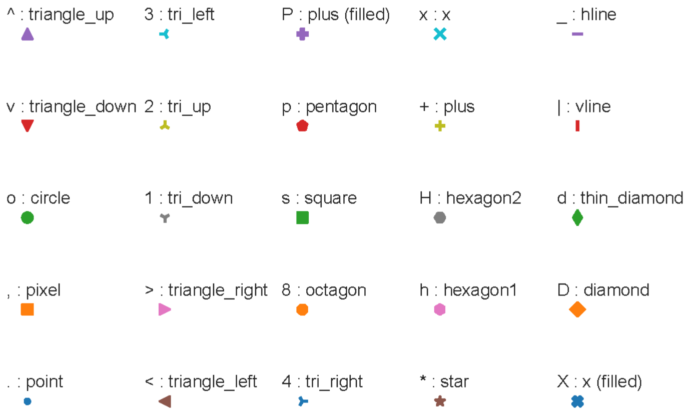
You can use the keyword argument marker to emphasize each point with a specified marker:

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o')
plt.show()
```



Matplotlib Markers

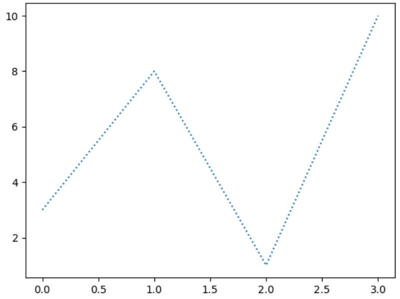
You can use the keyword argument marker to emphasize each point with a specified marker:



Matplotlib Line

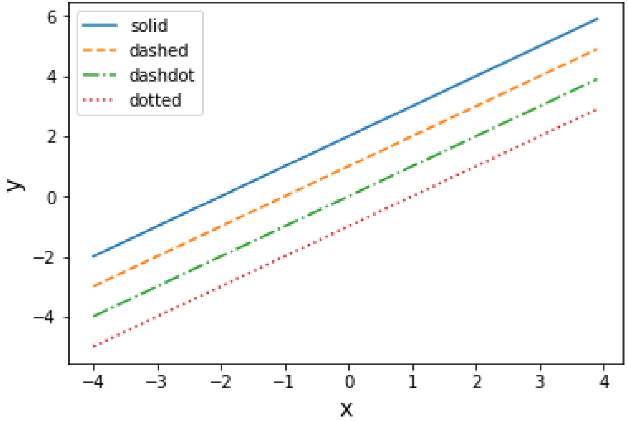
You can use the keyword argument linestyle, or shorter ls, to change the style of the plotted line:

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, linestyle = 'dotted')
plt.show()
```



Matplotlib Line

You can use the keyword argument linestyle, or shorter ls, to change the style of the plotted line:



Matplotlib Labels and Title

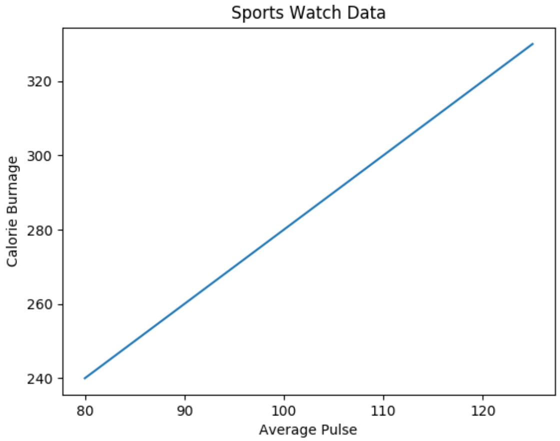
With Pyplot, you can use the xlabel() and ylabel() functions to set a label for the x- and y-axis. With Pyplot, you can use the title() function to set a title for the plot.

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
ypoints = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
plt.plot(xpoints, ypoints)
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.title("Sports Watch Data")
plt.show()
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Matplotlib Labels and Title

With Pyplot, you can use the xlabel() and ylabel() functions to set a label for the x- and y-axis. With Pyplot, you can use the title() function to set a title for the plot.



香 港 中 文 大 學 (深 圳)
The Chinese University of Hong Kong, Shenzhen

Multi-threading in Python: Threading

In Python, the **threading** module provides a simple and intuitive API for spawning multiple threads in a program. Let us consider an example using a threading module:

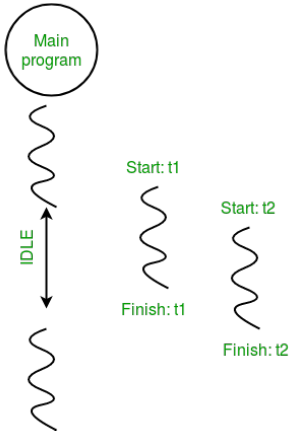
```
import threading


def print_cube(num):
    # function to print cube of given num
    print("Cube: {}" .format(num * num * num))


def print_square(num):
    # function to print square of given num
    print("Square: {}" .format(num * num))


if __name__ =="__main__":
    # creating thread
    t1 = threading.Thread(target=print_square, args=(10,))
    t2 = threading.Thread(target=print_cube, args=(10,))
```

```
# starting thread 1
t1.start()
# starting thread 2
t2.start()

# wait until thread 1 is completely executed
t1.join()
# wait until thread 2 is completely executed
t2.join()

# both threads completely executed
print("Done!")
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Multi-threading in Python: Threading

In Python, the **threading** module provides a simple and intuitive API for spawning multiple threads in a program. Let us consider an example using a threading module:



Multi-threading in Python: Thread Pool

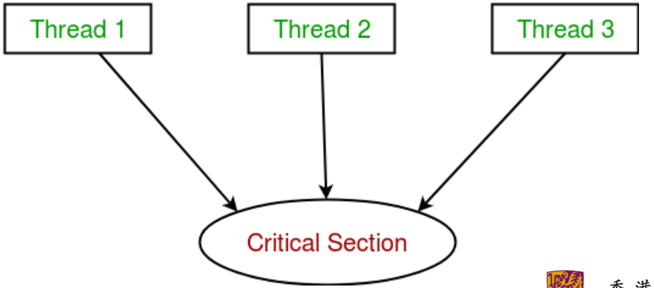
A **thread pool** is a collection of threads that are created in advance and can be reused to execute multiple tasks. The concurrent.futures module in Python provides a ThreadPoolExecutor class that makes it easy to create and manage a thread pool.

```
import concurrent.futures

def worker():
    print("Worker thread running")

# create a thread pool with 2 threads
pool = concurrent.futures.ThreadPoolExecutor(max_workers=2)

# submit tasks to the pool
pool.submit(worker)
pool.submit(worker)

# wait for all tasks to complete
pool.shutdown(wait=True)

print("Main thread continuing to run")
```

香 港 中 文 大 學 (深 圳)
The Chinese University of Hong Kong, Shenzhen

Multi-threading in Python: Synchronization

Thread synchronization is defined as a mechanism that ensures that two or more concurrent threads do not simultaneously execute some particular program segment known as the critical section.



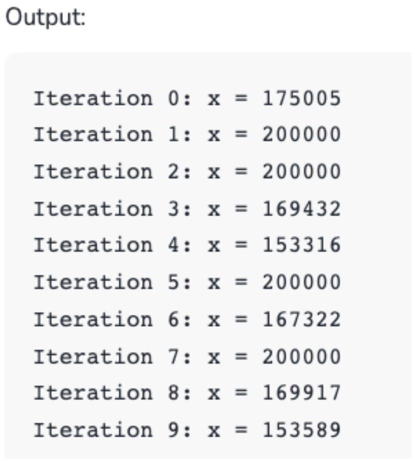
Multi-threading in Python: Synchronization

```python
import threading

# global variable x
x = 0

def increment():
    """
    function to increment global variable x
    """
    global x
    x += 1

def thread_task():
    """
    task for thread
    calls increment function 100000 times.
    """
    for _ in range(100000):
        increment()
```

```python
def main_task():
    global x
    # setting global variable x as 0
    x = 0

    # creating threads
    t1 = threading.Thread(target=thread_task)
    t2 = threading.Thread(target=thread_task)

    # start threads
    t1.start()
    t2.start()

    # wait until threads finish their job
    t1.join()
    t2.join()

if __name__ == "__main__":
    for i in range(10):
        main_task()
        print("Iteration {0}: x = {1}".format(i,x))
```

Multi-threading in Python: Synchronization

The expected final value of x is 200000 but what we get in 10 iterations of main_task function is some different values.

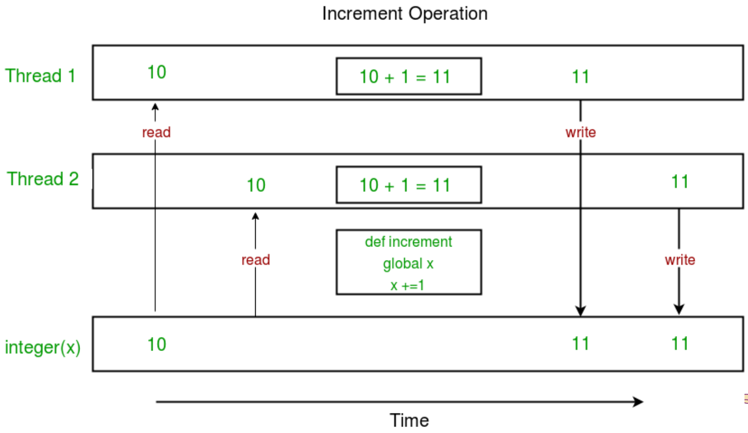

Output:

```
Iteration 0: x = 175005
Iteration 1: x = 200000
Iteration 2: x = 200000
Iteration 3: x = 169432
Iteration 4: x = 153316
Iteration 5: x = 200000
Iteration 6: x = 167322
Iteration 7: x = 200000
Iteration 8: x = 169917
Iteration 9: x = 153589
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Multi-threading in Python: Synchronization

This happens due to concurrent access of threads to the shared variable x. This unpredictability in the value of x is nothing but a race condition.



Multi-threading in Python: Synchronization

The threading module provides a **Lock class** to deal with the race conditions. The lock is implemented using a Semaphore object provided by the Operating System.

Lock class provides the following methods:

- **acquire([blocking])** : To acquire a lock. When invoked with the blocking argument set to True (the default), thread execution is blocked until the lock is unlocked, then the lock is set to locked and returns True.
- **release()** : To release a lock. When the lock is locked, reset it to unlocked, and return. If any other threads are blocked waiting for the lock to become unlocked, allow exactly one of them to proceed.

香 港 中 文 大 學 (深 圳)
The Chinese University of Hong Kong, Shenzhen

Multi-threading in Python: Synchronization

```python
import threading

# global variable x
x = 0

def increment():
    """
    function to increment global variable x
    """
    global x
    x += 1

def thread_task(lock):
    """
    task for thread
    calls increment function 100000 times.
    """
    for _ in range(100000):
        lock.acquire()
        increment()
        lock.release()
```

```python
def main_task():
    global x
    # setting global variable x as 0
    x = 0

    # creating a lock
    lock = threading.Lock()

    # creating threads
    t1 = threading.Thread(target=thread_task, args=(lock,))
    t2 = threading.Thread(target=thread_task, args=(lock,))

    # start threads
    t1.start()
    t2.start()

    # wait until threads finish their job
    t1.join()
    t2.join()
```

香 港 中 文 大 學 (深 圳)
The Chinese University of Hong Kong, Shenzhen

Multi-threading in Python: Synchronization

```python
if __name__ == "__main__":
    for i in range(10):
        main_task()
        print("Iteration {0}: x = {1}".format(i,x))
```

Output:

```
Iteration 0: x = 200000
Iteration 1: x = 200000
Iteration 2: x = 200000
Iteration 3: x = 200000
Iteration 4: x = 200000
Iteration 5: x = 200000
Iteration 6: x = 200000
Iteration 7: x = 200000
Iteration 8: x = 200000
Iteration 9: x = 200000
```

香 港 中 文 大 學 (深 圳)
The Chinese University of Hong Kong, Shenzhen

Question and Answering (Q&A)