AIoT Coding, Engineering and Entrepreneurial (AIoT CE^2) Skills Education for Gifted Students – Introduction to Python 2

Department of Electrical Engineering
香港城市大學
City University of Hong Kong

Table of Contents

Functions

- A function is a **block of codes** that will be run when it is called
- Data can be passed into and returned by a function
- Python function can be declared with the ***def*** keyword and called like this:

```python
#param and return value are optional
def func(param1, param2):
    #calculations here
    # ...
    return value1, value2
```

```python
val1, val2 = func(arg1, arg2)
```

- Data **being passed** into a function are called **arguments**
- **Variables being defined** in a function are called **parameters**
- A function can take multiple arguments and return multiple values

Functions

- Function without parameters nor return values is defined and called like this:

```python
def hello():
    print('hello world!')
    print('How are you?')
```

```python
hello()
```

```
hello world!
How are you?
```

Functions

- Why do we need to use functions?

  - ➢ Repeating same calculation for different inputs may be annoying
  - ➢ More likely to make an error

- Using functions makes program more efficient and readable!

Learn a programmer's motto: DRY – don't repeat yourself.

Example - Quadratic Equation

- A quadratic equation is an equation of the form:

$$ax^2 + bx + c = 0$$

where a, b, and c are the real number coefficients ($a,b,c \in \mathbb{R}$), and $a \neq 0$

- The roots $x_{1,2}$ can be found by the below equation:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- When $b^2-4ac > 0$, there are two real roots,
- When $b^2-4ac = 0$, there is one real root,
- When $b^2-4ac < 0$, there is no real roots

Example - Quadratic Equation

- Write a function to calculate the roots of the quadratic equations, accept the coefficients a, b, and c as the function arguments, determine whether b^2-4ac < 0, return None if so, otherwise return the two roots (two same roots if b^2-4ac = 0).

```python
def quadratic(a,b,c):
    delta = b**2 - 4*a*c
    if delta < 0:
        return None
    else:
        x1 = (-b+(delta)**0.5)/(2*a)
        x2 = (-b-(delta)**0.5)/(2*a)
        return x1, x2
```

$$2x^2 + 6x + 4 = 0$$

$$a = 2, b = 6, c = 4$$

```python
quadratic(2,6,4)
```

```
(-1.0, -2.0)
```

$$x^2 + 2x + 1 = 0$$

$$a = 1, b = 2, c = 1$$

```python
quadratic(1,2,1)
```

```
(-1.0, -1.0)
```

Scope of Variables

- Scope is the region where the variable name is valid, the function’s body is a local scope and elsewhere is the global scope, it can be used to classify local and global variables:

- Local variables
  - ➢ Defined and accessed only within a function
  - ➢ Once the function ends, the value attached to the local variable disappears
  - ➢ Variables defined inside a function are local by default

- Global variables
  - ➢ Defined outside of the function
  - ➢ Can be accessed and modified within a function with the ***global*** keyword

Examples for Global vs Local Variables

- a is defined locally within the func()
- Attempts in accessing it outside the func() will result in an error

```
def func():
    a = 2
    print('a within the func:', a)
func()
```

```
a within the func: 2
```

```
print('a outside the func:', a)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-719-3821ed63bd89> in <module>()
----> 1 print('a outside the func:', a)

NameError: name 'a' is not defined
```

Examples for Global vs Local Variables

```python
x = 1
```

```python
def func():
    x = 2
    print('x within the func:', x)

func()
print('x outside the func:', x)
```

```
x within the func: 2
x outside the func: 1
```

A new local variable x is created, it is valid only within the function (won't change the value of x outside the function).

```python
x = 1
```

```python
def func():
    global x
    x = 2
    print('x within the func:', x)

func()
print('x outside the func:', x)
```

```
x within the func: 2
x outside the func: 2
```

x is declared as global, the change of value of x is valid also outside the function

Exercise1

- The Eye Aspect Ratio (EAR) is used to estimate the level of openness of the eyes, which can be used for anti-spoofing in a face recognition system.

- Refer to the instructions in your jupyter notebook, write the functions to calculate the EAR.

Generators

- Generators are a special class of functions that return an iterator,
    - i.e. *iter(*iterable*)* is an example of an iterator
- Instead of ***return***, generators contains the ***yield*** keyword
- Example of a generator which reverse a string:

```
def reverse_str(n):
    for i in range(len(n)-1,-1,-1):
        yield n[i]
```

Generators

- The stream of values returned by generators can be processed using:
  - *next()*
  - Or for loop

```
rev_str = reverse_str('hi123')
print(next(rev_str), end='')
print(next(rev_str), end='')
print(next(rev_str), end='')
print(next(rev_str), end='')
print(next(rev_str), end='')
```

```
321ih
```

```
rev_str = reverse_str('hi123')
```

```
for i in rev_str:
    print(i, end='')
```

```
321ih
```

Example – Getting Bitcoin Price

- Normal functions get all the data at once
- Generators get the data on the fly

```
start_time = datetime.now()

symbol = 'BTCUSDT'
interval = '1m'
start = '01/07/2022 00:00:00'
end = '02/07/2022 23:59:59'

klines = client.get_historical_klines(
    symbol=symbol,
    interval=interval,
    start_str=start,
    end_str=end,
    klines_type=HistoricalKlinesType.FUTURES
)
print(datetime.now() - start_time)
```

```
0:00:21.019168
```

```
start_time = datetime.now()

symbol = 'BTCUSDT'
interval = '1m'
start = '01/07/2022 00:00:00'
end = '02/07/2022 23:59:59'

klines_generator = client.get_historical_klines_generator(
    symbol=symbol,
    interval=interval,
    start_str=start,
    end_str=end,
    klines_type=HistoricalKlinesType.FUTURES
)
print(datetime.now() - start_time)
```

```
0:00:00.000120
```

Recursion

- Recursive function is a function that calls itself
- Recursion:
  - Solving a problem by reducing it to a simpler problem of the same kind
  - Repeat this process until the problem is simple enough to be solved directly
  - The simplest problem is called the ***base case***
  - If the base case hasn't been reached, we perform the **recursive case**

Template for a recursive function:

```
def recursiveFunc(param):
    if stopping_condition: //base case
        //do something
        return solution
    else:  //recursive case
        //do something
    return recursiveFunc(modified_param)
```

Recursion – Factorial

Definition of the factorial for a non-negative integer *n:*

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{if } n > 0 \end{cases}$$

Base case

Recursive case

Recursion – Factorial

Code:

```
def factorial(n):
    #Base case for invalid n
    if n<0 or n%1 != 0:
        return 'Please enter a non negative integer!'
    #Base case for n=0
    elif n==0:
        return 1
    #Recursive case
    else:
        return n*factorial(n-1)
```

Output:

```
print(factorial(-10))
print(factorial(2.1))
print(factorial(0))
print(factorial(3))
print(factorial(10))
```

```
Please enter a non negative integer!
Please enter a non negative integer!
1
6
3628800
```

Exercise2 - Power Set

The power set of an input set contains the sets of every possible combination of numbers inside the input set (including the empty set). For example, the power set of the input set {1,2,3} is:
{ {}, {1}, {2}, {3}, {1,2}, {2,3}, {1,3}, {1,2,3} }.

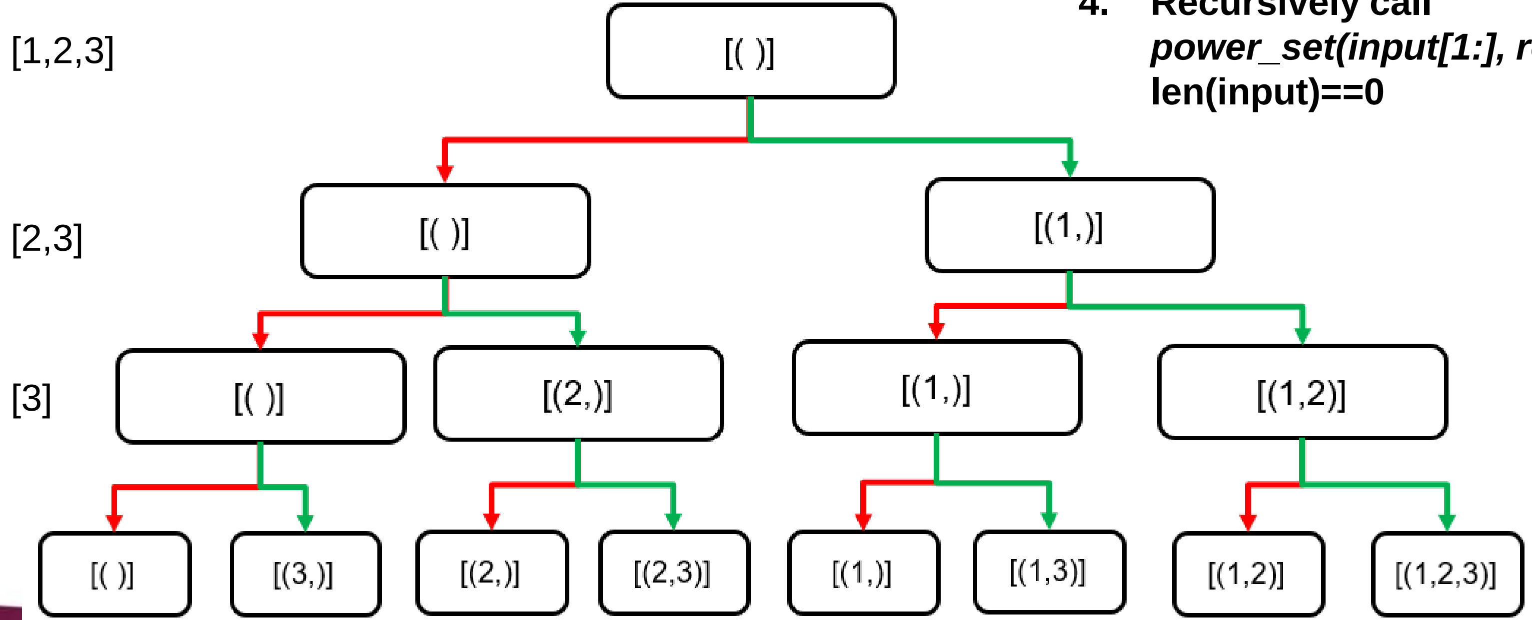
1. Write a function in Python such that given an input set, return the power set.

2. Can you modify your function for the power set such that it returns the nCr combinations given the input set (all possible combinations of choosing r items out of n items in the input set)?

Exercise2 - Power Set

1. Initialize the list *result* = [()]
2. Declare a function called *power_set(input, result)*, where *input* is the input set
3. At each call to *power_set()*, iterate through *result*, add *input[0]* to each element of *result* and add it to *result*
4. Recursively call *power_set(input[1:], result)* until len(input)==0

input = [1,2,3]



Object-Oriented Programming in Python

- Object-oriented programming (OOP) is a programming paradigm
- An object-oriented program is made up of classes and objects which are like real-world objects, i.e. an object of a class called *Human*
- An object contains: **data attributes** and **methods**

   - **Data attributes:** data associated with the object,
     i.e. name, age, gender, etc

   - **Methods:** functions associated with the object, to specify its behaviors,
     i.e. singing, dancing, etc can be the behaviors of the *Human* class

Class and Object

- Class
  - A class is a blueprint for objects
  - Specify the data attributes and methods (behaviors) an object should have
  - Methods are defined inside the body of a class to define the behaviors of its objects
  - For example, *Human* class can have the singing and dancing methods

- Object
  - An object is an instance of a class
  - For example, we can create an object called Carrie from the *Human* class

Class Declaration

```python
class Human:

    #Class variables
    species = 'mammal'

    #Initializer
    def __init__(self, name, age, gender):
        #Instance variables
        self.name = name
        self.age = age
        self.gender = gender

    #Method
    def sing(self, song):
        return '{} is singing the song {}'.format(self.name, song)
    def dance(self):
        return '{} is dancing'.format(self.name)
```

- A class is defined using the ***class*** keyword
- A class may define a special method named *__init__()* to create objects customized to a specific initial state
- **Class variables:**
  - Share by all instances
- **Instance variables:**
  - Unique to each instance
- All methods, including *__init__(),* should have the *self* parameter

Object Creation and Calling Methods

```python
# Create a Human object
carrie = Human('Carrie', 18, 'Female')

#Access the instance variables using dot operator
print('The age of {} is {}'
      .format(carrie.name, carrie.age))

#Call the methods using dot operator
print(carrie.sing('Yesterday'))
print(carrie.dance())
```

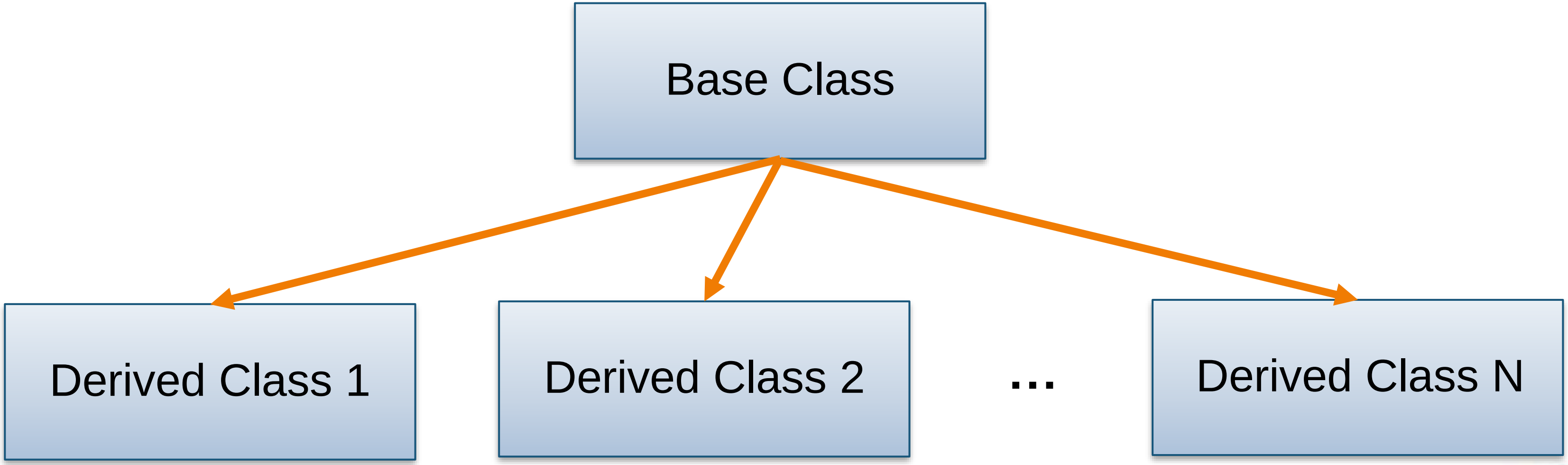
```
The age of Carrie is 18
Carrie is singing the song Yesterday
Carrie is dancing
```

- The values for the instance variables can be initialized when creating a *Human* object

- The data attributes and methods can be accessed with the dot operator

Inheritance

- Inheritance allows us to define a derived class from a base class
- Which inherit all the attributes and methods from the base class



Inheritance

- All classes in Python inherits from the *object* class
- *issubclass(class_A,class_B)* returns True if *class_A* inherit from *class_B*
- For example, to check whether the *Human* class inherit from *object:*

```
print(issubclass(Human,object))
```

```
True
```

- *isinstance(obj,class_A)* returns True if *obj* is an object created from *class_A* or its derived class
- For example, bool is derived from int:

```
print(isinstance(1, int))
print(isinstance(True, int))
```

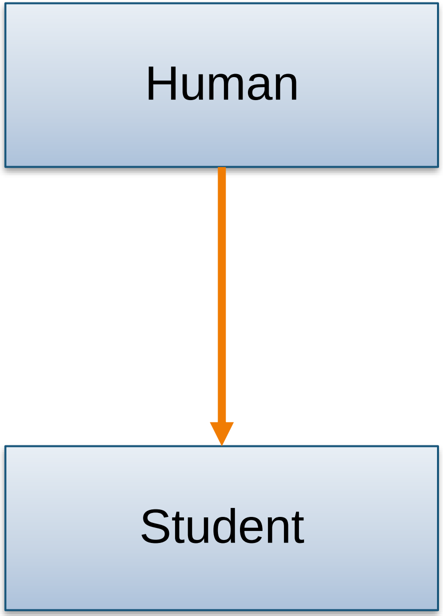
```
True
True
```

Inheritance

- We can create a class *Student* which is derived from *Human*, and add a method called *study()*:

```python
class Student(Human):

    def study(self, subject):
        return '{} is studying {}'.format(self.name, subject)
```



Inheritance

- The *Student* class inherit all the attributes and methods from the *Human* class
- The *study()* method only available for objects created from the Student class

```python
mary = Student('Mary',15,'Female')
print('The age of {} is {}'
      .format(mary.name, mary.age))
print(mary.sing('Yesterday'))
print(mary.dance())
print(mary.study('computer science'))
```

```
The age of Mary is 15
Mary is singing the song Yesterday
Mary is dancing
Mary is studying computer science
```

```python
print(carrie.__class__)
print(carrie.study('computer science'))
```

```
<class '__main__.Human'>

---------------------------------------------------------------------------
AttributeError                            Traceback (mos
<ipython-input-31-3e3650c504e3> in <module>
      1 print(carrie.__class__)
----> 2 print(carrie.study('computer science'))

AttributeError: 'Human' object has no attribute 'study'
```

Method Overriding and super()

- We can override the methods of the base class by defining a method with the same name in the derived class
- *super()* can be used to access methods from the base class, which are being overridden by the derived class

```
class Student(Human):

    def __init__(self, name, age, gender, yr_of_study):
        super().__init__(name, age, gender)
        self.yr_of_study = yr_of_study

    def study(self, subject):
        return '{} is studying {}'.format(self.name, subject)

    def sing(self, song, grade):
        return super().sing(song) + ' in the music exam, and {} got grade {}'.format(self.name, grade)
```

Method Overriding and super()

- The *Student* class now has an extra instance variable *yr_of_study* after overriding the *__init__()* method of its base class

- The *Student.sing()* method is different from the *Human.sing()* method

```
mary = Student('Mary',15,'Female','F5')
print('{} is now in {}'.format(mary.name, mary.yr_of_study))
print(mary.sing('Yesterday','A'))
```

```
Mary is now in F5
Mary is singing the song Yesterday in the music exam, and Mary got grade A
```

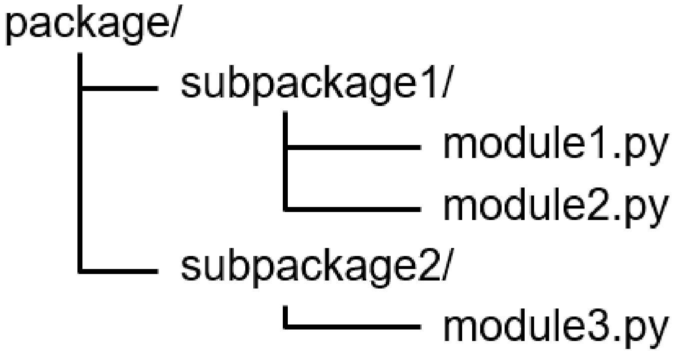
Exercise3

- Refer to your jupyter notebook write a class for *rectangle*. In your *rectangle* class, there are two instance variables called *height* and *width.* You should also implement the methods *get_perimeter()* and *get_area()* which calculate the perimeter and the area of the rectangle respectively.

- Write a class for *square*, your *square* class should be a derived class of *rectangle* and should only have one instance variable called *length*.

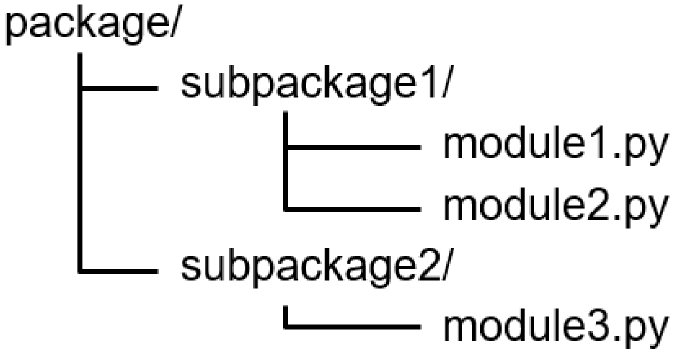
Python Library

- Python definitions and statements (i.e. functions, classes) can be saved in a .py file, and can be imported in other python program files.
- The .py file is called **a module**
- Different modules can be put into a directory with a *__init__.py* file, called **a package**
- Sometimes a package can contain subpackages
- The package can be published as **a library**



Python Library

- A package can be imported with the ***import*** keyword (usually at the top of your program)
- Modules and functions in a package can be accessed with ‘**.**’ operator
- The package can be renamed inside your program with the ***as*** keyword
- For example, to use func1() in module1.py:
  - ➢ import package as p
  - ➢ x = p.subpackage1.module1.func1(param1, param2)



Python Library

- If you only want to import func1() and func2() from module1:
  - ➢ from package.subpackage1.module1 import func1, func2
- If you want to import every functions from module1, you can use ‘ ***** ’
  - ➢ from package.subpackage1.module1 import *

```
package/
    ├── subpackage1/
    │       ├── module1.py
    │       └── module2.py
    └── subpackage2/
            └── module3.py
```

Pre-Installed Packages in Anaconda

- Many useful packages (i.e. numpy, pandas, sqlite etc) are pre-installed in Anaconda
- You can check the installed packages by typing the following command in Anaconda Prompt or terminal:
  - ➢ conda list

```
(base) C:\Users\[redacted]>conda list
# packages in environment at C:\Users\[redacted]\Anaconda3:
#
# Name                    Version                   Build  Channel
_ipyw_jlab_nb_ext_conf    0.1.0            py36he6757f0_0
absl-py                   0.2.2                     <pip>
alabaster                 0.7.10           py36hcd07829_0
anaconda-client           1.6.14                   py36_0
anaconda-navigator        1.8.7                    py36_0
anaconda-project          0.8.2            py36hfad2e28_0
asn1crypto                0.24.0                   py36_0
astor                     0.6.2                     <pip>
astroid                   1.6.1                    py36_0
astropy                   2.0.3            py36hfa6e2cd_0
```

Managing Environments Using Conda

- Some applications may depend on a specific version of python or packages
- For example:
  - Application A requires: python3.7.6 and NumPy1.19.4
  - Application B requires: python3.8.3 and NumPy1.20.2
- Impossible for one Python installation to meet the requirements for all applications
- Solution: Create a virtual environment for the specific application

Managing Environments Using Conda

- Conda is a package and environment manager
- Included in all versions of Anaconda and Miniconda
- To create a new environment for Python:

➢ **Option1** Run the command: *conda create --name env_name python*

Example:

Anaconda Prompt (Anaconda3)

```
(base) C:\Users\[redacted]>conda create --name aiot python
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: D:\Anaconda3\envs\aiot

  added / updated specs:
    - python
```

Managing Environments Using Conda

- **Option2** Create an environment.yml file and run the command:

  conda env create -f environment.yml

  Example:

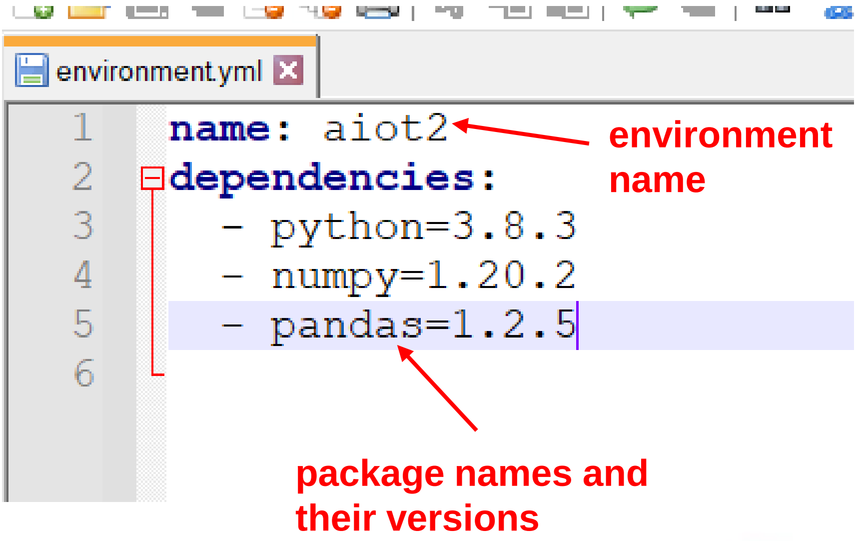
Anaconda Prompt (Anaconda3)

```
(base) C:\Users\[redacted]>conda env create -f environment.yml
Collecting package metadata (repodata.json): done
Solving environment: done

Downloading and Extracting Packages
numpy-1.20.2         | 23 KB     | ############################################################################ | 100%
mkl_fft-1.3.0        | 137 KB    | ############################################################################ | 100%
python-3.8.3         | 15.6 MB   | ############################################################################ | 100%
vc-14.2              | 8 KB      | ############################################################################ | 100%
pip-21.1.3           | 1.8 MB    | ############################################################################ | 100%
wincertstore-0.2     | 15 KB     | ############################################################################ | 100%
certifi-2021.5.30    | 140 KB    | ############################################################################ | 100%
mkl_random-1.2.1     | 223 KB    | ############################################################################ | 100%
mkl-service-2.4.0    | 51 KB     | ############################################################################ | 100%
pandas-1.2.5         | 7.9 MB    | ############################################################################ | 100%
numpy-base-1.20.2    | 4.2 MB    | ############################################################################ | 100%
setuptools-52.0.0    | 726 KB    | ############################################################################ | 100%
Preparing transaction: done
Verifying transaction: done
```

Managing Environments Using Conda

- The environment.yml file specifies the environment name and package dependencies
- This file can be shared with others:
  - ➢ Other developers can reproduce your environment easily
  - ➢ Ensure your result is reproducible on other developers' machine



Creating an environment file manually documentation: https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html#create-env-file-manually

Managing Environments Using Conda

- Activate the environment using the command *conda activate env_name*
- The installed packages of the new environment will be different from the base environment (check it by the *conda list* command!)

Changed from base env to aiot env

```
(base) C:\Users\[redacted]>conda activate aiot

(aiot) C:\Users\[redacted]>conda list
# packages in environment at D:\Anaconda3\envs\aiot:
#
# Name                    Version                   Build  Channel
ca-certificates           2021.7.5             haa95532_1
certifi                   2021.5.30        py39haa95532_0
openssl                   1.1.1k               h2bbff1b_0
pip                       21.1.3           py39haa95532_0
python                    3.9.5                h6244533_3
setuptools                52.0.0           py39haa95532_0
sqlite                    3.36.0               h2bbff1b_0
tzdata                    2021a                h52ac0ba_0
vc                        14.2                 h21ff451_1
vs2015_runtime            14.27.29016          h5e58377_2
wheel                     0.36.2             pyhd3eb1b0_0
wincertstore              0.2              py39h2bbff1b_0
```

Different installed packages for different env

Installing Python Packages

- To install new Python packages:
  - conda install package_name=version no.(optional)
  - or pip install package_name ==version no.(optional)
- Difference between conda install and pip install:
  - https://www.anaconda.com/blog/understanding-conda-and-pip

	conda	pip
Package type	any	Python only
Create environment?	Yes	No, requires virtualenv or venv
Dependency checks?	Yes	No
Package sources	Anaconda Repository/Cloud	Python Package Index (PyPI)

Installing Python Packages

Anaconda Prompt - conda install scrapy

```
(base) C:\Users\      >conda install scrapy
Solving environment: done

## Package Plan ##

  environment location: C:\Users\      \Anaconda3

  added / updated specs:
    - scrapy


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    pyasn1-modules-0.2.2       |           py36_0          86 KB
    pytest-runner-4.2          |           py36_0          12 KB
    scrapy-1.5.1               |           py36_0         329 KB
    hyperlink-18.0.0           |           py36_0          62 KB
    conda-4.5.11               |           py36_0         1.0 MB
    pyasn1-0.4.4               |   py36h28b3542_0         101 KB
    zope-1.0                   |           py36_1           4 KB
    automat-0.7.0              |           py36_0          70 KB
    zope.interface-4.5.0       |   py36hfa6e2cd_0         203 KB
    parsel-1.5.0               |           py36_0          28 KB
    service_identity-17.0.0    |   py36h28b3542_0          18 KB
    queuelib-1.5.0             |           py36_0          21 KB
    twisted-18.7.0             |   py36hfa6e2cd_1         4.9 MB
    appdirs-1.4.3              |   py36h28b3542_0          16 KB
```

Anaconda automatically hands the package dependency for you

Installing Python Packages

Anaconda Prompt

```
The following packages will be UPDATED:

    conda:          4.5.4-py36_0          --> 4.5.11-py36_0

Proceed ([y]/n)? y

Downloading and Extracting Packages
pyasn1-modules-0.2.2 |   86 KB | ############################################################################ | 100%
pytest-runner-4.2    |   12 KB | ############################################################################ | 100%
scrapy-1.5.1         |  329 KB | ############################################################################ | 100%
hyperlink-18.0.0     |   62 KB | ############################################################################ | 100%
conda-4.5.11         |  1.0 MB | ############################################################################ | 100%
pyasn1-0.4.4         |  101 KB | ############################################################################ | 100%
zope-1.0             |    4 KB | ############################################################################ | 100%
automat-0.7.0        |   70 KB | ############################################################################ | 100%
zope.interface-4.5.0 |  203 KB | ############################################################################ | 100%
parsel-1.5.0         |   28 KB | ############################################################################ | 100%
service_identity-17. |   18 KB | ############################################################################ | 100%
queuelib-1.5.0       |   21 KB | ############################################################################ | 100%
twisted-18.7.0       |  4.9 MB | ############################################################################ | 100%
appdirs-1.4.3        |   16 KB | ############################################################################ | 100%
constantly-15.1.0    |   13 KB | ############################################################################ | 100%
cssselect-1.0.3      |   28 KB | ############################################################################ | 100%
incremental-17.5.0   |   25 KB | ############################################################################ | 100%
w3lib-1.19.0         |   31 KB | ############################################################################ | 100%
pydispatcher-2.0.5   |   19 KB | ############################################################################ | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Type “y” to continue

CityU

Conda Command Cheatsheet

- Create a new environment for Python
  - conda create --name env_name python=version no. (optional)
- Activate an environment
  - conda activate env_name
- Deactivate an environment
  - conda deactivate
- See a list of all your environments
  - conda info –envs
- Update a package to the latest compatible version
  - conda update package_name
- Remove a package from the activate environment
  - conda remove package_name

More conda commands:
https://docs.conda.io/projects/conda/en/latest/commands.html

Next Lesson…

Advanced Python
NumPy
- NumPy array
- Array creation, shapes and arithmetic
- Array broadcasting and broadcasting rules **Exercise**
- Optimization problems and coding the Newton–Raphson algorithm in Python

專業 創新 胸懷全球
Professional · Creative
For The World

Department of
Electrical Engineering
香港城市大學
City University of Hong Kong