Tutorial 4: Python for Beginners

Zhizhen Ye (Euclid)

My Information

- YE Zhizhen (Euclid)
  - Office:
    - SHB 1005
  - Office Hours:
    - Wednesday 9:30am – 10:30am
  - zzye@cse.cuhk.edu.hk

Outline

- Programming environment
- Basic properties of Python
- Classes and objects
- Assignment 2 preview
- Learning resources

Python

- Python programming language
  - is an interpreted language
  - is dynamic, object-oriented, general-purpose
- Special features:
  - Dynamic type system and automatic memory management
  - Duck Typing

Python Installation

- Download here
- Python 2.7 for assignment 2

Version	Operating System
Gzipped source tarball	Source release
XZ compressed source tarball	Source release
Mac OS X 32-bit i386/PPC installer	Mac OS X
Mac OS X 64-bit/32-bit installer	Mac OS X
Windows debug information files	Windows
Windows debug information files for 64-bit binaries	Windows
Windows help file	Windows
Windows x86-64 MSI installer	Windows
Windows x86 MSI installer	Windows

Running the first Python program

- Open any text editor, e.g. Notepad++, Sublime Text
- Type in the following code

```
1 print("hello world")
~
~
```

- Save it as “hello.py” in your home folder

Running the first Python program

- Windows users: Open the Command Prompt
- Mac users: Open the Terminal
- Type “python hello.py”

```
Euclid:/uac/gds/zzye/3180ta/python$ python hello.py
hello world
```

Python IDE (optional)

- PyCharm
- https://www.jetbrains.com/pycharm/download/

Dynamic typing

- Don’t need to declare variables or their data type
- Can change the data type of a variable
- Example:
  - Python code:

```
container = "Hello world!"
print(container)
container = 3180
print(container)
```

  - Output:

```
Hello world!
3180
```

String Formatting

- Python uses C-style string formatting

```
name = "Peter"
age = 18
```

"%" operator to format variables

a tuple of variables

```
print("%s is %d years old" % (name, age))
# Peter is 18 years old
```

```
gpa = 3.809
print("His GPA is %.2f." % gpa)
# His GPA is 3.81.
```

Indentation

- Uses indentation for blocks, instead of curly braces
- Can use tab or 4 spaces for indentation
- e.g., If-Then-Else Blocks

Python code:

```python
if returnval == 1:
    result = "Success"
    print(result)
elif returnval == 0:
    result = "Failure"
    print(result)
else:
    result = "Unknown"
```

Java code:

```java
if(returnval == 1){
    result = "Success";
    System.out.println(result);
}
else if(returnval == 0){
    result = "Failure";
    System.out.println(result);
}
else{
    result = "Unknown";
}
```

List

- Python's built-in data structure
- Written within square brackets [ ]
- Can store any type of variable
- Can use len() to return the length of a list

```
aList = ["Jimmy", "Isaac", 10]
print(aList[0])      # Jimmy
print(aList[2])      # 10
print(len(aList))    # 3
```

List initialization

- range(n) generates [0, 1, 2, …, n-1]
- e.g. range(5) is [0, 1, 2, 3, 4]

- range(a,b) generates [a, a+1, a+2, …, b-1]
- e.g. range(3, 9) is [3, 4, 5, 6, 7, 8]

- [v] * n generates a list of v with a length of n
- e.g. [0] * 3 is [0, 0, 0]

List methods

- The size of list can be changed dynamically
- list.append()
  - Add an element at the end of a list
- list.pop()
  - Remove and return the last element of a list
- list.pop(i)
  - Remove and return list[i]

```
aList = [1,3,5]
aList.append(7)
print aList          # [1,3,5,7]
print aList.pop(1)   # 3
print aList          # [1,5,7]
```

2D list

- A list with another list

```
drinks = ["coke", "water", "beer"]
snacks = ["chocolate", "peanut"]
shoppingCart = [drinks, snacks]
print shoppingCart
#[['coke', 'water', 'beer'], ['chocolate', 'peanut']]
print shoppingCart[0]
#['coke', 'water', 'beer']
print shoppingCart[0][2]
#beer
```

- Can be used to represent a game board

For loop

- The "for element in list" is used to iterate over a list
- e.g., sum up all the numbers in a list

```
primes = [1,2,3,5,7]
total = 0
for num in primes:
    total += num
print(total)    # 18

sum(primes)     # 18
```

For loop

- Traditional numerical for loop in python:
  - for i in range(n):
- e.g., print the numbers from 0 through 4

Python code:

```
for i in range(5):
    print i
```

Java code:

```
for(int i = 0; i < 5; i++){
    System.out.println(i);
}
```

While loop

- For example, print out 0 to 5

```
i = 0
while i < 5:
    print i
    i += 1
```

- break and continue are also supported in while loop and for loop

```
i = 0
while i < 5:
    if i % 2 == 0:
        continue
    print i
    i += 1
```

Define a class

class name

For unification purpose. Explained in next tutorial

```
class WaterTypePokemon(object):
```

Define a class

```
class WaterTypePokemon(object):
        def __init__(self, HP, Weight):
```

Define a method

class constructor

The first argument to each method is *self*

Define a class

```
class WaterTypePokemon(object):
        def __init__(self, HP, Weight):
            self.HP = HP
            self.weight = Weight
```

Define attributes

Self is the object itself
Equivalent to *this* in Java

Define a class

```
class WaterTypePokemon(object):
        def __init__(self, HP, Weight):
                self.HP = HP
                self.weight = Weight
        def Swim(self):
                print("It is swimming!")
```

Define another method

Instantiate an object

```
class WaterTypePokemon(object):
        def __init__(self, HP, Weight):
                self.HP = HP
                self.weight = Weight
        def Swim(self):
                print("It is swimming!")

pkm1 = WaterTypePokemon(80, 30)
```

Instantiate a WaterTypePokemon object

Only need to pass the HP and Weight argument
Python will add the *self* argument for you

Instantiate an object

```
class WaterTypePokemon(object):
        def __init__(self, HP, Weight):
                self.HP = HP
                self.weight = Weight
        def Swim(self):
                print("It is swimming!")

pkm1 = WaterTypePokemon (80, 30)
pkm1.HP = 90
```

Access an object's attribute

Instantiate an object

```
class WaterTypePokemon(object):
        def __init__(self, HP, Weight):
                self.HP = HP
                self.weight = Weight
        def Swim(self):
                print("It is swimming!")

pkm1 = WaterTypePokemon(80, 30)
pkm1.HP = 90
pkm1.Swim()
```

Call an object's method

Inheritance

class Psyduck(WaterTypePokemon):

Put its super class into a bracket
Psyduck extends WaterTypePokemon class
Inherit the attributes and methods of its super class

Inheritance

```
class Psyduck(WaterTypePokemon):
        def __init__(self, HP, Weight, HairCount):
```

Override its superclass's constructor

Inheritance

```
class Psyduck(WaterTypePokemon):
        def __init__(self, HP, Weight, HairCount):
                super(Psyduck, self).__init__(HP, Weight)
```

Call its superclass's constructor to initialize the HP and Weight attributes

Inheritance

```
class Psyduck(WaterTypePokemon):
        def __init__(self, HP, Weight, HairCount):
                super(Psyduck, self).__init__(HP, Weight)
                self.hairCount = HairCount
```

Initialize the HairCount attribute

Inheritance

```
class Psyduck(WaterTypePokemon):
        def __init__(self, HP, Weight, HairCount):
                super(Psyduck, self).__init__(HP, Weight)
                self.hairCount = HairCount
        def Scratch(self, target):
                target.HP -= 5
```

Define another method

Inheritance

```
class Squirtle(WaterTypePokemon):
    def WaterGun(self, target):
        target.HP -= 10
```

We do not define a constructor for the Squirtle class
It will inherit the constructor of its superclass

Example

v.s.

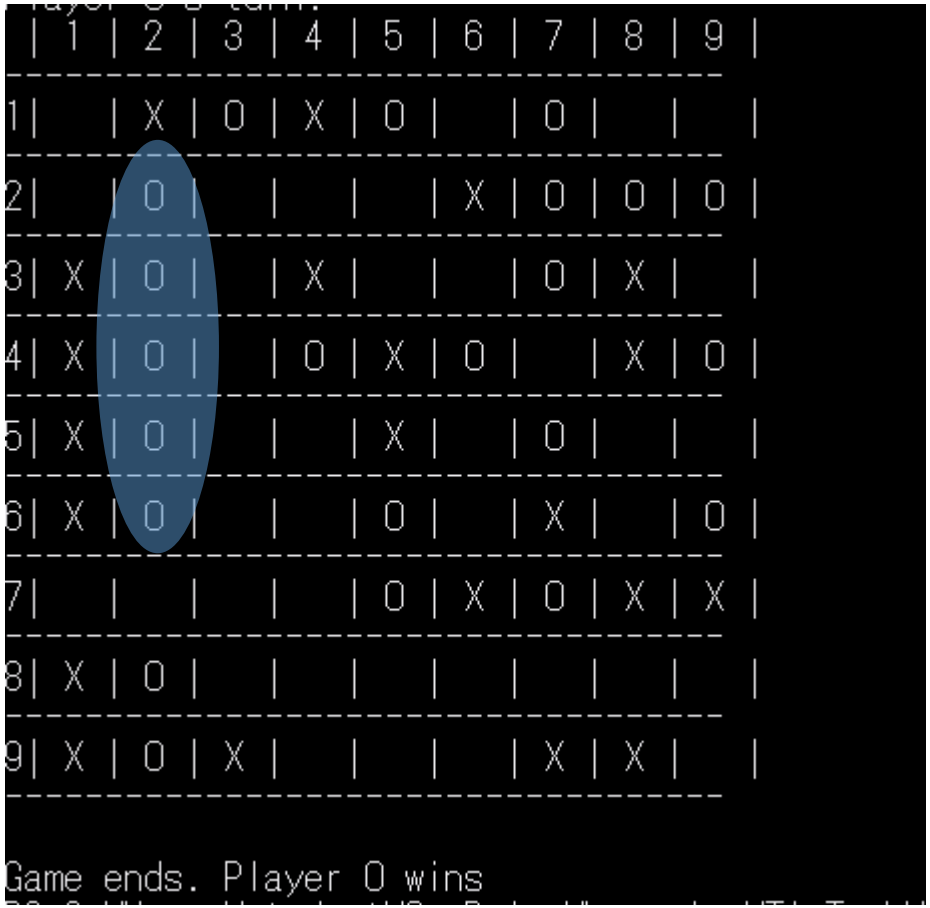
- A simple battle between a Psyduck and Squirtle

```
psyduckA = Psyduck(100, 30, 3)
squirtleA = Squirtle(100, 25)
while psyduckA.HP > 0 and squirtleA.HP > 0:
    psyduckA.Scratch(squirtleA)
    squirtleA.WaterGun(psyduckA)
    print "psyduckA's HP: %d" % (HP)
    print "squirtleA's HP: %d" % (HP)

if psyduckA.HP > 0:
    print "psyduckA wins!"
elif squirtleA.HP > 0:
    print "squirtleA wins!"
else:
    print "Draw game!"
```

Assg.2 Task 1: Gomoku

- Two players: player o and player x



```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
-----------------------------------
1|   | X | O | X | O |   | O |   |   |
-----------------------------------
2|   | O |   |   |   | X | O | O | O |
-----------------------------------
3| X | O |   | X |   |   | O | X |   |
-----------------------------------
4| X | O |   | O | X | O |   | X | O |
-----------------------------------
5| X | O |   |   | X |   | O |   |   |
-----------------------------------
6| X | O |   |   | O |   | X |   | O |
-----------------------------------
7|   |   |   |   | O | X | O | X | X |
-----------------------------------
8| X | O |   |   |   |   |   |   |   |
-----------------------------------
9| X | O | X |   |   |   | X | X |   |
-----------------------------------

Game ends. Player O wins
```

Assg.2 Task 1: Gomoku

- Use Python 2.7
- Follow the OOP design:
  - Implement all the methods specified in the specification
  - Allowed to add extra functions/methods

Assg.2 Task 2

- State the advantages of Dynamic Typing
- Give sample code to demonstrate each advantage

Learning resources

- Official tutorial website:
- https://docs.python.org/2/tutorial/
- Documentation website:
- https://docs.python.org/2/index.html
- Interactive Python tutorial:
- https://www.codecademy.com/learn/learn-python
- Duck Typing:
- https://en.wikipedia.org/wiki/Duck_typing

END

- Q&A