python™

sodas @ NTU CSIE Mobile & HCI Research Lab

http://www.ntumobile.org/

Python is an interpreted, general-purpose high-level programming language whose design philosophy emphasizes code readability.

Python aims to combine "remarkable power with very clear syntax", and its standard library is large and comprehensive. Its use of indentation for block delimiters is unique among popular programming languages.

Python2.x vs. Python3.x

- Print is a function, but not a statement.
- Fully supported unicode
- Views and iterators instead of lists

Declare Variables

Python uses **duck typing** and has typed objects but untyped variable names.

Despite being **dynamically typed**, Python is **strongly typed**, *forbidding operations that are not well-defined* rather than silently attempting to make sense of them.

Python uses **reference** like **Java** to operate object. Objects are strong typed, but the reference *itself* is **not typed**!

```
>>> a="A"

>>> print(a.__class__)

<type 'str'>

>>> a=1

>>> print(a.__class__)

<type 'int'>
```

Variable “a” is a reference to a string object in line 1.

You can use “__class__ “ method to check its type.

Variable “a” becomes a reference to an integer object in line 4.

“Reference” is not typed, so you can change it quickly. But the object it referenced is typed.

IDLE Console output

is **vs.** ==(equal operator)

- “is” is used to identify object. More clearly, this operator will compare the memory references.
- “==” is used to compare object. That is, this operator checks the content of objects.

```
>>> a = [1]
>>> b = [1]
>>> a is b
False
>>> a == b
True
```

Variable “a” and “b” are two list object. (Python List is like ArrayList in Java)

Since “a” and “b” are 2 different object in memory, “is” operator returns false.

But “a” and “b” have the same content exactly, “==” operator returns true.

More clearly, “is” works like “==” in other language, and “==” works like “equals” in other one.

IDLE Console output

- “str” *(Python string object)* supports string interning like Java, so “is” will return true.
  (See example next page)*(String interning also called string pool)*

- “None” *(Python Null)* is point to the same memory reference, so no matter “is” or “==” return the same result.

```
>>> a="a"

>>> b=str("a")

>>> a is b

True

>>> a == b

True
```

Unlike Java, no matter how you create the string object, the string object with same value will share the same memory space by string interning.

IDLE Console output

Indent Block Statement

Or called “suite” in python

You must use **4 spaces** to do indent suite but not use 1 tab to do so.

So, you can set your IDE convert your tab input into 4 spaces.

```
if (a==NULL) {
    if (b==1) {
        printf(“Yeah”);
    }
    printf(“XD”);
}
```

```
if a is not None:
____if b==1:
________print(“Yeah”)
____print(“XD”)
```

C vs. Python

```c
int add(int a, int b) {
    printf(“Add now”);
    return a+b;
}
// Run the function
int result = add(1,2);
```

```python
def add(a, b):
    print(“Add now”)
    return a+b;
# Run the function
result = add(1,2);
```

C vs. Python

String Operation

The power of Python

```
>>> vowel = "AEIOU"
>>> pattern = "Apple Computer, Inc."
>>> vowelCount = 0
>>> for alphabet in pattern:
    if alphabet.upper() in vowel:
        vowelCount = vowelCount + 1
>>> print(vowelCount)
6
```

You can use “in” as quick enumerate to get each element (or alphabet) in string object.

And also use “in” operation to check if this object is in a list. String in python is also like a list object.

Next page will show another string operation in python.

```
>>> sentence = "A cat is sleeping."

>>> print(sentence[2])

c

>>> print(sentence[0:5])

A cat

>>> print(sentence[9:])

sleeping.
```

Since string object could act like list object, you can use “[]” to access some element inside it.

In python, we use “:” to represent range. For example, “0:5” means from the first one to 6th one. (From index=0 to index=5)

If one side of “:” leaves empty, it means to the end. (Or from the start). For instance, “[9:]” means from 10th to the end, and “[:3]” means from start to 3rd.

```
>>> print("ABCDE"[-1])
E
>>> print("ABCDE"[:-2])
ABC
>>> print("ABCDE"[-2:])
DE
>>> print("ABCDE"[2:-2])
C
```

Negative number means count from back.

So “[-1]” means the last one of “ABCDE”.

“[:-2]” means from start to the last 3rd one, and “[2:-2]” is from 3rd to last 3rd.

Finally, it’s easy to join two strings. Just “add” them together like Java.

- Note that in Python 2.x, it's terrible that string object doesn't support unicode. There's another class: "unicode" which save content in unicode format.

- You should save Chinese character with unicode object, and be careful for using list-like operation to deal with Chinese string.

- In Python 3.x, all string object support unicode directly. There's no unicode class in 3.x. So you can just save Chinese character in string object.

Data Structure

List, Tuple, and Dictionary

- A **List** is like an *ArrayList* in Java. List is mutable class in Python.

- **Tuple** is another ArrayList-like object, but it’s immutable.

- **Dictionary** is a data structure like *HashMap* in Java. It’s a key-value object.

```
>>> list1 = [1, "B", 3]
>>> list1 = list1 + ["D"]
>>> print(list1)
[1, 'B', 3, 'D']
>>> list2 = list1[0:2]
>>> print(list2)
[1, 'B']
```

You can join two list via “+” operator and access element or part of list via “[]” or “[a:b]”.

```
>>> list1 = [5,3,6,1,4,2]
>>> list2 = list1
>>> list1 is list2
True
>>> list1.sort()
>>> list1 is list2
True
>>> print(list2)
[1, 2, 3, 4, 5, 6]
```

An example of object reference in Python.

Since we assign “list1” to “list2”, “list2” will reference the same object of “list1”.

Now “list1” changed (sort), and the “list2” is also changed.

Also, since list is mutable data structure, it still have the same reference after sorted.

```
>>> tuple1 = (1,3,2)
>>> tuple1.sort()

Traceback (most recent call last):
  File "<pyshell#51>", line 1, in <module>
    tuple1.sort()
AttributeError: 'tuple' object has no attribute 'sort'
```

Tuple is immutable, so you can't sort (modify) it.

Once if you want to sort your tuple, you have to copy it, modify it, and restore it.

```
>>> dict1 = { "name": "Mike", "gender": "Male" }
>>> dict1["job"] = "professor"
>>> print(dict1)
{'gender': 'Male',
 'job': 'professor',
 'name': 'Mike'}
>>> print(dict1["name"])
Mike
>>> dict1["name"] = "Mike Chen"
>>> dict1.pop("job")
'professor'
>>> print(dict1)
{'gender': 'Male', 'name': 'Mike Chen'}
```

A dictionary is made by "{}" syntax. If you typed "{}", you'll get an empty dictionary.

You can add key-value pair in to a dictionary like line 2. Just define key and assign value directly.

Get or update value is quiet easy. You can use "pop" to remove pair in dictionary.

Control Statement

```
>>> a=1
>>> b=2
>>>
if a == b:
    print(“Equal”)
elif a>b:
    print(“a is bigger”)
else:
    print(“Not equal”)
Not equal
```

Here, the code left is not correct for IDLE (why?), but it could make you more clearly.

Left example is If-elif-else statement. Note that Python uses “elif”.

```
>>> list1 = [1,2,3,4,5]
>>> result = 0
>>>
for integer in list1:
   result = result + integer
>>> print(result)
15
>>>
for i in range(1,3):
    print(i)
1
2
```

Code left contains some IDLE bugs, too.

“for” loop can use in 2 ways: quick enumerate and “range”

```
>>>
try:
    doSomething()
except:
    print("QQ")
>>> a = 0
>>>
while a < 10:
    print(".")
    a = a+1
```

Code left contains some IDLE bugs, too.

Exception handler in Python is "try-except".

Function/Method define

```
>>>
def add(a, b):
    return a+b;
>>>
def sayHello():
    print("Hello")
>>> print(add(1,2))
3
>>> sayHello()
Hello
```

Code left contains some IDLE bugs, too.

Just use “def” to implement your function. Do not have to declare the type of this function, too.

It’s very free to use return or not, since Python is duck-typed.

OOP

```
class Fraction (object):
    # Member
    numerator=0
    ...
    # Method
    # Constructor
    def __init__(self, n, d):
        ...
    # get float value
    def getFloatValue(self):
        …
    # get float value
    def add(self, a, b):
        ...
```

Python class supports multiple-inheritance and also override operator like "+".

Every method in class must define at least one argument - "self" which means class itself.

While calling method of object, you should not call the "self".

For example, "half" is an instance of "Fraction" class. To call methods, you just have to type:
half.getFloatValue() not half.getFloatValue(self).
half.add(F1, F2) not half.add(self, F1, F2).

```
  F1 = Fraction(1,3)
F1.getFloatValue()
```

You don't have to type "new" to alloc object. Python did that for you. Just call constructor (__init__) directly.

Methods/Variables named with "__" prefix and postfix are Python internal methods/variables. So it's better that do not name your method/variable with "__".

For example, "__init__" is preserved by Python as object constructor.

There's no private/public/protected modifier in Python class.

aptana

Pydev

PYTHON DEVELOPMENT ENVIRONMENT

Let’s open Eclipse with Pydev

The attached source file is a Pydev project. *Just import it to Eclipse/Pydev*.
It contains some example of Python, including OO.