0701 Python / AI Programming Practice

Tutor: Nanyang Ye
Note Taker: Y. Qiu

Fuction Programming

map(function, iterable, ...)

底层实现：for → require iterable data structure (e.g. list)

In [4]:

```
num = list(map(int,input().split()))
print(num)

string = list(map(str,input().split()))
string
```

```
10 20 30
[10, 20, 30]
zhao qian sun li
```

Out[4]:

```
['zhao', 'qian', 'sun', 'li']
```

Notice:

"map" and "input" sometimes may conflict with each other.
A better example is as follows.

In [18]:

```
num = list(map(int, [1,2,3.1]))
num
```

Out[18]:

```
[1, 2, 3]
```

reduce(function, iterable)

apply function to all elements in iterable data structure

In [1]:

```
num = list(map(int,input().split()))

def mul(a,b):
    return a*b
# end mul

from functools import reduce
result = reduce(mul,num)
result
```

```
10 20 30
```

Out[1]:

```
6000
```

filter(function, iterable)

```
    e.g. Filter out all NaN (Not a Number) or INF (infinite numbers)
```

In [23]:

```
numlist = list(map(int,input().split()))

def less(x):
    return (x<=65)
# end less

num = list(filter(less, numlist))
print(num)

num1 = list(filter(lambda x: x<=65, numlist))
print(num1)
```

```
1 2 34 66 7 18
[1, 2, 34, 7, 18]
[1, 2, 34, 7, 18]
```

sorted()

```
    usable parameters: key, reverse
```

In [16]:

```
a = [1,4,3,7,5,8989,-5]
print(sorted(a))

from functools import cmp_to_key

def cmp(x,y):
    if x*x<y*y:      return -1
    elif x*x == y*y: return 0
    else:            return 1
# end cmp

print(sorted(a,key = cmp_to_key(cmp), reverse=True))   # reverse = True: ↓
```

```
[-5, 1, 3, 4, 5, 7, 8989]
[8989, 7, 5, -5, 4, 3, 1]
```

Nested Functions (Utilizing Annonymous Functions)

In [40]:

```
def convert(num):
    def value(bit):
        if bit>='0' and bit<='9' : return eval(bit)
        else:                      return ord(bit)-87
    # end value

    def convert(num, base):
        sum = value(num[0])
        for i in range(1,len(num)):
            sum = sum*base + value(num[i])
        return sum
    # end convert

    if (num[1] == 'b'):    return convert(num[2:],2)        # binary
    elif (num[1] == 'x'):  return convert(num[2:],16)       # hexadecimal
    elif (num[1] == 'o'):  return convert(num[2:],8)        # octal
    else:                  return num                       # decimal

    return result
# end convert

print(convert(input()))
```

```
0x6242f
402479
```

In [25]:

```
add = lambda x: x+1
do_it_three_times = lambda f: lambda x: f(f(f(x)))    # lambda f is a function whose only paramete
print(do_it_three_times(do_it_three_times)(add)(0))

# do_it_three_times(add)(x):  add(add(add(x))), i.e. x+3
# do_it_three_times(do_it_three_times)(add)(x), i.e. do_it_three_times(add)(0) → 3,
#                                                    do_it_three_times(add)(3) → 9,  do_it_three_ti
```

```
27
```

Function as Return Value

Decorators

In [22]:

```
import functools
def log(func):             # the input is a function!
    @functools.wraps(func)
    def wrapper(*val, **kwargs):
        print('call %s{}' % func.__name__)       #.__name__ (double underline!) : return the name of
        return func(*val, **kwargs)
    return wrapper

@log
def test(x):
    print(x)

test(1)
```

```
call test{}
1
```

Object-Oriented

Class

```
class ClassName:
    'Help Information'          # can be automatically
    class body
```

Initialization Function:

```
def __init__(parameters):               # "__<name>__" implies that the function is a pri
vate function
```

Notice:

```
In python, when defining member functions, there must at least be one parameter, a.k.a. "self".
"self" is similar to (or exactly) "*this" in C++ and C.
However, "self" is not a pointer, thus "self.<>" is correct while "self-><>" is incorrect.
```

Application member functions:

```
<>.function(parameters)
function(<>,parameters)
Both acceptable.
```

Changeable and Unchangeable Objects

```
Changeable Object: Only transfer the value (formal parameters, 形参)
    number(int/float/...), string, tuple
Unchangeable Object:  the parameter in the function is the object itself (real parameters/quotations, 实参/引用)
    list, dictionary, set
```

In [27]:

```python
def ChangeList(a):
    a.append(6)
    print("In function:",a)

x = [1,2,3,4]
print(x)
ChangeList(x)
print(x)
```

```
[1, 2, 3, 4]
In function: [1, 2, 3, 4, 6]
[1, 2, 3, 4, 6]
```

In [48]:

```python
def ChangeInt(a):
    a = 100000
    print("In function:",a)

x = 12
print(x)
ChangeInt(x)
print(x)
```

```
12
In function: 100000
12
```

In [47]:

```
def ChangeInt(a):
    a = 100000
    print("In function:",a)
    return a

x = 12
print(x)
x = ChangeInt(x)
print(x)
```

```
12
In function: 100000
100000
```

Private Members

```
xx: public members
_xx : protected members (can be visited outside the class in the same file where it is defined)
__xx: private members   (cannot be visited outside the class)
```

Inherit

```
base class, derived class

class <Derived_Name>(Base_Name):
    class body

Also inherit functions in the base class, no need to redefine.
Functions in the derived class will automatically disable the funcitons of the same name in the base class.

In fact, "self" parameters is used to distinguish functions of the same name of the base and derived classes.
```

In [84]:

```
class base:
    def __init__(self,number=0):
        self.number = number

    def ParentMethod(self):
        print("调用了父类方法")

    def Method(self):
        print("同名方法：父类实现")

    def __ParentPrivate__(self):
        print("私有")

class derived(base):
    def __init__(self,number=0):
        self.number = number

    def ChildMethod(self):
        print("调用了子类方法")

    def Method(self):
        print("同名方法：子类实现")

p = base()
c = derived()

p.ParentMethod()
c.ChildMethod()
c.ParentMethod()          # Derived classes can use functions in its base class.

p.Method()
c.Method()

p.__ParentPrivate__()
print(p.__ParentPrivate__.__name__)
```

```
调用了父类方法
调用了子类方法
调用了父类方法
同名方法：父类实现
同名方法：子类实现
私有
__ParentPrivate__
```

Visual Class

Decrators of the visual class requires all derived classes to redefine the method functions.

```
import abc
class <name>(metaclass = abc.ABCMeta):
    @abc.abstractmethod
    def <abstract_function>:
        pass       # python require at least one meaningful line for a funtion. "pass" means the function is an abstract one.
                    # "pass" means "do nothing"
```

Multi-States (Utilizing Visual Class)

In [39]:

```python
import abc
class Animal(metaclass = abc.ABCMeta):
    @abc.abstractmethod
    def talk(self):
        pass

class Cat(Animal):
    def talk(self):
        print("Meow.")

class Human(Animal):
    def talk(self):
        print("Hello.")

class Dog(Animal):
    def talk(self):
        print("Woof.")


def talk(obj):
    obj.talk()
    return

h = Human()
c = Cat()
d = Dog()

talk(h)
talk(c)
talk(d)
```

```
Hello.
Meow.
Woof.
```

绑定关系

默认绑定到对象。

绑定到类：@classmethod 装饰，类似C++的静态成员

An Example

In [57]:

```
class Student:
    number = 0

    def __init__(self,name,age,score):
        self.name  = name
        self.age   = age
        self.score = score
        Student.number += 1           # static member of the class, regarded as a property of the cl

    def get_name(self):
        return self.name

    def get_age(self):
        return self.age

    def get_course(self):
        maxcourse = self.score[0]
        for i in self.score:
            if maxcourse<i:
                maxcourse = i
        return maxcourse

# Test

a = Student("ZhangMing",20,[69,88,100])
print(a.get_name())
print(a.get_age())
print(a.get_course())
```

```
ZhangMing
20
100
```

File

File Input and Output

```
file_object = open(file_name, access_mode = r, buffering = 0)
access_mode:
    r    -- read only
    rb   -- read only as binary
    r(b)+ -- read and write (as binary). File pointer at the beginning.
    w    -- write only. Overwrite the file if exists.
    wb   -- write only as binary
    a    -- append
buffering: 是否使用缓存

file_object:
    name
    mode
    closed [BOOL]
    close

<file>.write()
```

In [55]:

```
file = open("1.txt","w")
file.write("Hey there.\t What a day.\r")
file.write("Whatever, this is just an example.\n")
file.write("The file will be closed soon.\n")
file.close()
```

In [58]:

```
f = open("1.txt","r")
string = f.read()
f.close()

words = string.split()              # space(" "), table("\t"), enter("\r") and linefeed("\n") will be
print("There are {0} words in the file.".format(len(words)))
print(words)

f = open("1.txt","r")
string = f.readline()               # only a line
f.close()

words = string.split()
print("There are {0} words in the file.".format(len(words)))
print(words)
```

```
There are 17 words in the file.
['Hey', 'there.', 'What', 'a', 'day.', 'Whatever,', 'this', 'is', 'just', 'an', 'exa
mple.', 'The', 'file', 'will', 'be', 'closed', 'soon.']
There are 5 words in the file.
['Hey', 'there.', 'What', 'a', 'day.']
```

Exception（异常）

Passive Exception

```
try:
    #{block}
except<(error1, error2)<as e(can be omitted)>>:
    #{solve it}
except error:
    #{solve it}
except:  #{solve it}
```

Exception

```
raise Name_of_the_Error("Information of the error.")
```

Finally

```
No matter there is an error or not, run the line after "finally:".
Often used to release the space or other sources.
```

Else

```
When there is no error in the block, run "else:<line>"
```

With

```
for line in open("1.txt","r"):
    print(line,end="")
# Problem: cannot close the file (the file is created as an annonymous class)

with open("1.txt) as f:
    for line in f:
        print(line,end="\n")
# When the "with" block is over, f is automatically closed and all sources are released.
```

Pass

```
do nothing.
```

In [62]:

```
def add_to_list_in_dict(thedict, listname, element):
    try:
        l = thedict[listname]
        print("%s already has %d elements." % (listname, len(l)))
    except:
        thedict[listname] = []
        print("Create %s." % listname)
    finally:
        thedict[listname].append(element)
        print("Added %s to %s." % (element, listname))

a = dict()
add_to_list_in_dict(a, "A", "Abandon")
add_to_list_in_dict(a, "A", "Abnormal")
add_to_list_in_dict(a, "B", "Balloon")
add_to_list_in_dict(a, "Z", "Zoo")
add_to_list_in_dict(a, "B", "Basket")
add_to_list_in_dict(a, "A", "Apple")
```

```
Create A.
Added Abandon to A.
A already has 1 elements.
Added Abnormal to A.
Create B.
Added Balloon to B.
Create Z.
Added Zoo to Z.
B already has 1 elements.
Added Basket to B.
A already has 2 elements.
Added Apple to A.
```

Std Modules and Third-Party Modules

Modules

```
numbers, math, cmath, decimal, random

isinstance(number, type) = True/False
```

In [65]:

```
c = 1j+2
print(type(c))

a = complex(1,2)
print(a)

isinstance(a,int)
```

```
<class 'complex'>
(1+2j)
```

Out[65]:

```
False
```

In [70]:

```
def isint(x):
    try:
        return isinstance(eval(x),int)
    except:
        return False

numlist = list(filter(isint,input().split()))
print(numlist)
```

```
1 23 j+4 5.4 as 7
['1', '23', '7']
```

In [89]:

```
import math
a, b, c = 17.0, 2.7, -5.3
print(math.ceil(a),math.ceil(b),math.ceil(c))
a, b, c = math.floor(a),math.floor(b),math.floor(c)
print(a,b,c)
a = math.factorial(a)
print(a)
print(math.log10(a))
```

```
17 3 -5
17 2 -6
355687428096000
14.5510685151576
```

In [95]:

```
import cmath
print(cmath.exp(2.7))
print(cmath.exp(2.8+0.7j))
```

```
(14.879731724872837+0j)
(12.577559605526668+10.593932310316903j)
```

In [71]:

```
import random
for i in range(10):
    x = random.randrange(1,1000)
    print(x)
```

```
483
578
446
789
894
368
282
687
789
984
```

In [100]:

```
import decimal
print(decimal.Decimal.from_float(21.220))
print(decimal.Decimal('212.34134213164412312').quantize(decimal.Decimal('0.00')))
print(decimal.Decimal('212.34134213164412312').quantize(decimal.Decimal('1.007')))
```

```
21.219999999999998863131622783839702606201171875
212.34
212.341
```

In [80]:

```
from functools import partial


def convert(num, base):
    def value(bit):
        if bit>='0' and bit<='9' : return eval(bit)
        else:                      return ord(bit)-87
    sum = value(num[0])
    for i in range(1,len(num)):
        sum = sum*base + value(num[i])
    return sum

binaryconvert = partial(convert,base=2)
n = input()
print(binaryconvert(n))
```

```
111
7
```

pathlib / os.path / fileinput /

```
    fileinput.input("<Menu>/*.txt")        iterable data structure
```

pickle

```
pickle.dump(data,file)
pickle.load(file)
```

In [90]:

```
import pickle as pkl
with open("2.txt","wb") as f:
    data = 10
    pkl.dump(data,f)

with open("2.txt","rb") as f:
    data = pkl.load(f)
    print(data)
```

```
10
```

os / io / time / logging

threading / multiprocessing