Python 教程学习笔记

通过之前的学习，我们已经对 Python 了解了一些。现在，根据官网文档我们继续补充一些 Python 的知识。

代码流的控制

type

```
print(type(1))
```

```
<class 'int'>
```

```
print(type('a'))
```

```
<class 'str'>
```

type 函数很有用，来打印对象的类型。

range

range 函数是非常很有用的。

```
for i in range(5):
  print(i, end = ' ')
```

```
0 1 2 3 4
```

```
for i in range(2, 6, 2):
  print(i, end = ' ')
```

```
2 4
```

看 range 函数的定义。

```
class range(Sequence[int]):
    start: int
    stop: int
    step: int
```

可见是一个类。

```
print(range(5))
```

```
range(0, 5)
```

而不是：

```
[0,1,2,3,4]
```

继续。

```
print(list(range(5)))
```

```
[0, 1, 2, 3, 4]
```

为什么。看 `list` 的定义。

```
class list(MutableSequence[_T], Generic[_T]):
```

list 的定义是 `list(MutableSequence[_T], Generic[_T]):`。而 `range` 的定义是 `class range(Sequence[int])`。list 继承了 `MutableSequence`。range 继承了 `Sequence`。

继续往下找是这样的。

```
Sequence = _alias(collections.abc.Sequence, 1)
MutableSequence = _alias(collections.abc.MutableSequence, 1)
```

这里我们不明白它俩的关系。但大概我们知道了为什么可以这样写 `list(range(5))`。

函数参数

来看函数的补充知识。

```
def fn(a = 3):
  print(a)

fn()
```

```
3
```

这是给参数一个默认值。

```
def fn(end: int, start = 1):
  i = start
  s = 0
  while i < end:
    s += i
    i += 1
  return s

print(fn(10))
```

```
45
```

end 是必须要有的参数。注意到要把必须要有的参数写在最前面。

```
def fn(start = 1, end: int):

    def fn(start = 1, end: int):
                               ^

SyntaxError: non-default argument follows default argument
```

注意到 end 是 non-default argument。start 是 default argument。意思是说，非默认参数跟在了默认参数后面。就是说必须把非默认参数放在所有默认参数前面。start 是默认参数，即是如果不传递的话，默认已经有值了。

```
def fn(a, /, b):
  print(a + b)


fn(1, 3)
```

这里/来把参数类型分隔。有两种形式的传递参数方式。一种是靠位置来传递，一种是靠指定关键词来传递。

```
def fn(a, /, b):
  print(a + b)


fn(a=1, 3)

    fn(a=1, 3)
              ^

SyntaxError: positional argument follows keyword argument
```

这样写就不行。a=1 表示这是靠关键词来传递参数。这把它当做了一个关键词参数。而 b 则是位置参数。

```
def f(pos1, pos2, /, pos_or_kwd, *, kwd1, kwd2):
      -----------    ----------     ----------
        |             |                  |
        |        Positional or keyword   |
        |                                - Keyword only
         -- Positional only
```

注意这里定义函数时，用上/和 * 已经隐含了各参数的传递类型。所以得按规则传递。

```
def fn(a, /, b):
  print(a + b)


fn(1, b=3)
```

上面这样就没报错。

```
def fn(a, /, b, *, c):
  print(a + b + c)


fn(1, 3, 4)
```

```
    fn(1, 3, 4)
TypeError: fn() takes 2 positional arguments but 3 were given
```

fn 只能接收 2 个位置参数，但是给了 3 个。

```
def fn(a, /, b, *, c):
  print(a + b + c)


fn(a = 1, b=3, c=4)
```

```
    fn(a = 1, b=3, c=4)
TypeError: fn() got some positional-only arguments passed as keyword arguments: 'a'
```

fn 有一些参数只能靠位置传递的现在则是用关键词来传递。

映射形式的参数

```
def fn(**kwds):
  print(kwds)


fn(**{'a': 1})
```

```
{'a': 1}
```

```
def fn(**kwds):
  print(kwds['a'])


d = {'a': 1}
fn(**d)
```

```
1
```

可见 ** 就是把参数展开。

```
def fn(a, **kwds):
  print(kwds['a'])
```

```
d = {'a': 1}
fn(1, **d)
```

```
TypeError: fn() got multiple values for argument 'a'
```

像 `fn(1, **d)` 这样调用函数时，展开就是 `fn(a=1, a=1)`。所以会出错。

```
def fn(**kwds):
  print(kwds['a'])

d = {'a': 1}
fn(d)
```

```
TypeError: fn() takes 0 positional arguments but 1 was given
```

如果像 `fn(d)` 这样调用函数，会被当成是位置参数，而不是展开成关键词参数。

```
def fn(a, / , **kwds):
  print(kwds['a'])

d = {'a': 1}
fn(1, **d)
```

这样却行。说明位置参数和映射形式的参数可以同名。

```
def fn(a, / , a):
  print(a)

d = {'a': 1}
fn(1, **d)
```

```
SyntaxError: duplicate argument 'a' in function definition
```

这样就出错了。注意这几种情况的微妙关系。

```
def fn(a, / , **kwds):
  print(kwds['a'])

fn(1, **[1,2])
```

```
TypeError: __main__.fn() argument after ** must be a mapping, not list
```

** 后面必须跟着映射。

可迭代类型的参数

```
def fn(*kwds):
  print(kwds)

fn(*[1,2])
```

```
(1, 2)
```

```
def fn(*kwds):
  print(kwds)

fn(*1)
```

```
TypeError: __main__.fn() argument after * must be an iterable, not int
```

`*` 必须跟着 `iterable`。

```
def fn(a, *kwds):
  print(type(kwds))

fn(1, *[1])
```

```
<class 'tuple'>
```

打印一下类型。这也是为什么上面输出 `(1,2)`，而不是 `[1,2]`。

```
def fn(*kwds):
  print(kwds)

fn(1, *[1])
```

```
(1, 1)
```

注意到这里调用 `fn(1, *[1])` 时，就把参数展开了，成了 `fn(1,1)`。然后在 `fn(*kwds)` 解析的时候，`kwds` 又把 `1,1` 变成了元组 `(1,1)`。

```
def concat(*args, sep='/'):
  return sep.join(args)

print(concat('a','b','c', sep=','))
```

```
a,b,c
```

Lambda 表达式

lambda 就是把函数当做变量来保存。还记得在「解谜计算机科学」一文中所说的吗。

```
def incrementor(n):
  return lambda x: x + n

f = incrementor(2)
print(f(3))
```

```
5
```

再看一个例子。

```
pairs = [(1, 4), (2,1), (0, 3)]

pairs.sort(key = lambda pair: pair[1])

print(pairs)
```

```
[(2, 1), (0, 3), (1, 4)]
```

```
pairs = [(1, 4), (2,1), (0, 3)]

pairs.sort(key = lambda pair: pair[0])

print(pairs)
```

```
[(0, 3), (1, 4), (2, 1)]
```

pair[0] 时，就按第一个数排序。pair[1] 时，就按第二个数排序。

文档注释

```
def add():
  """add something
  """
  pass

print(add.__doc__)
```

```
add something
```

函数签名

```
def add(a:int, b:int) -> int:
  print(add.__annotations__)
  return a+b

add(1, 2)
```

```
{'a': <class 'int'>, 'b': <class 'int'>, 'return': <class 'int'>}
```

数据结构

列表

```
a = [1,2,3,4]

a.append(5)
print(a)   # [1, 2, 3, 4, 5]

a[len(a):] = [6]
print(a)   # [1, 2, 3, 4, 5, 6]

a[3:] = [6]
print(a)   # [1, 2, 3, 6]

a.insert(0, -1)
print(a)   # [-1, 1, 2, 3, 6]

a.remove(1)
print(a)   # [-1, 2, 3, 6]

a.pop()
print(a)  # [-1, 2, 3]

a.clear()
print(a)  # []

a[:] = [1, 2]
print(a.count(1)) # 1
```

```
a.reverse()
print(a)   # [2, 1]

b = a.copy()
a[0] = 10
print(b)   # [2, 1]
print(a)   # [10, 1]

b = a
a[0] = 3
print(b)  # [3, 1]
print(a)  # [3, 1]
```

列表构造

```
print(3 ** 2)   # 9
print(3 ** 3)   # 27
```

先来学一种运算，**。这表示次方的意思。

```
sq = []
for x in range(10):
  sq.append(x ** 2)

print(sq)
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

接着试试用 map。

```
a = map(lambda x:x, range(10))
print(a)
# <map object at 0x103bb0550>
print(list(a))
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

sq = map(lambda x: x ** 2, range(10))
print(list(sq))
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

sq = [x ** 2 for x in range(10)]
```

```
print(sq)
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

可见 `for` 是很灵活的。

```
a = [i for i in range(5)]
print(a)
# [0, 1, 2, 3, 4]

a = [i+j for i in range(3) for j in range(3)]
print(a)
# [0, 1, 2, 1, 2, 3, 2, 3, 4]

a = [i for i in range(5) if i % 2 == 0]
print(a)
# [0, 2, 4]

a = [(i,i) for i in range(3)]
print(a)
# [(0, 0), (1, 1), (2, 2)]
```

嵌套列表构造

```
matrix = [[(i+j*4) for i in range(4)] for j in range(3)]
print(matrix)
# [[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]

t = []
for j in range(3):
  t.append([(i+j*4) for i in range(4)])
print(t)
# [[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]
```

注意这两段代码的方式。即是说：

```
[[(i+j*4) for i in range(4)] for j in range(3)]
```

也就相当于：

```
for j in range(3):
   [(i+j*4) for i in range(4)]
```

也即相当于:

```
for j in range(3):
  for i in range(4):
      (i+j*4)
```

所以这方便用来做矩阵转置。

```
matrix = [[(i+j*4) for i in range(4)] for j in range(3)]
print(matrix)
#  [[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]

mt = [[row[j] for row in matrix] for j in range(4)]
print(mt)
# [[0, 4, 8], [1, 5, 9], [2, 6, 10], [3, 7, 11]]

print(list(zip(*matrix)))
[(0, 4, 8), (1, 5, 9), (2, 6, 10), (3, 7, 11)]
```

del

```
a = [1, 2, 3, 4]

del a[1]
print(a)  # [1, 3, 4]

del a[0:2]
print(a) # [4]

del a
print(a) # NameError: name 'a' is not defined
```

字典

```
ages = {'li': 19, 'wang': 28, 'he' : 7}
for name, age in ages.items():
    print(name, age)

# li 19
# wang 28
```

```
# he 7

for name in ages:
    print(name)

# li
# wang
# he

for name, age in ages:
     print(name)

ValueError: too many values to unpack (expected 2)

for i, name in enumerate(['li', 'wang', 'he']):
    print(i, name)

# 0 li
# 1 wang
# 2 he

print(reversed([1, 2, 3]))
# <list_reverseiterator object at 0x10701ffd0>

print(list(reversed([1, 2, 3])))
# [3, 2, 1]
```

模块

脚本方式调用模块

```
import sys

def f(n):
    if n < 2:
        return n
    else:
        return f(n-1) + f(n-2)
```

```
if __name__ == "__main__":
    r = f(int(sys.argv[1]))
    print(r)
```

```
% python fib.py 3
2
```

```
% python -m fib 5
5
```

dir

```
import fib


print(dir(fib))
```

```
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec_
```

```
import builtins
print(dir(builtins))
```

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipe
```

包

包，即 pakages。

```
pk.py
fibp
   cal
       cal.py
   pt
       pt.py
```

cal.py:

```
def f(n):
    if n < 2:
        return n
    else:
        return f(n-1) + f(n-2)
```

```
def fl(n):
    return list(map(f, range(5)))
```

pt.py:

```
def p(l):
    print(l, end=' ')

def pln(l):
    print(l)
```

pk.py:

```
import fibp.cal.cal
import fibp.pt.pt

fibp.pt.pt.p(fibp.cal.cal.fl(10))
```

pk.py 也可以写成这样:

```
from fibp.cal import cal
from fibp.pt import pt

pt.p(cal.fl(10))
```