**immutable** (Cannot change): int, float, bool, string, tuple

**mutable**: list, set, dict

**iteratable**: list, tuple, string, dict, set

**indexable** (support get element by a[i]): list, string, tuple, dict

**hashable** (keep id until lifetime end): int, float, string, tuple

**bool false**: False, None, 0, 0.0, 0j, “” (empty string), [] (empty list) {} (empty dict), range(0), set() (empty set)

**Operators:**

> x // y -> floored quotient of x and y (3 // 2 = 1)

> is not -> negated object identity id(a) != id(b)

**Namespace:**

> *Built-in*: built-in names, print, int, NameError

> *Global*: global variables

> *Enclosed*: for variables in inside function (wrapped in a function)

> *Local*: For variables in functions

> Each imported module has its own namespace, parallel with local

**Import function:**

> **import math**: import whole math class, when use objects need to type math.sin()

> **from math import sin**: import sin object, when use only type sin()

**Positional argument**: number and order of the argument is important

**Keyword argument:** order is not important **func(x=2, z='123', y=1234)**

**Default/optional argument:** can be omitted, has default value

**Pure function**: function without side-effect (I/O task); only mapping; output depend only on input (can not use global variables)

**while(x<10):** when x<10, keep running. Once x>=10, stop

Function can read global variable directly, modify global variable by **global x**

**Generator Function**: return is changed to **yield**. **yield** will pause the function, keep the state and resume when the next calling (the value of variable will keep)

**Recursive**: Calling itself, solve smaller problem (divide & conquer), more running time for function calling. Often use DP instead of recursive

**Iterative**: for or while loop, faster

**Inner function:** can read global and outer function variable. Can modify global variable by **global x**, can modify **nearest enclosing namespace (outer function)** by **nonlocal x**

**Errors:** syntax, arithmetic. undeclared variable, incompatible types, increect numbers of args, infinite loop, numerical imprecision, logic

**String**:

> Immutable

> 'b' in 'banana' -> True

> len('hi') -> 2

> chr(123) -> ‘{’ (Unicode to character)

> ord('{') -> 123 (Character to ASCII)

> ‘string’[0::1] -> First number: start (inclusive), Second number: end (exclusive), Third number: interval(skip no. of character-1)

> 'IT5001'[0::3] -> I0 (Start from ‘I’, skip two)

> 'IT5001'[-1] -> 1 (last one)

> 'IT5001'[1:3:1] -> T50 (Start from the second, end at the third, no skip)

> 'I' not in 'IT5001' -> False (Case sensitive)

**List:**

> Mutable, can modify the element. Dynamic arrays.

> Can contain one or more types in a list, defined using []

> Can be sorted. **sort()** returns a sorted list, **sorted()** modify the origin to sorted.

> Can be reversed. **reverse()**

> **a[i],** return i-th element of a. **a[i:j]**, returns elements i up to j-1. **len(a),** **min(a),** **max(a).** **x in a** return True if x is a part of a. **a+b**, concatenates a and b. **n\*a**, creates n copies of sequence a. **Also, for tuple**

> append: **a.append()** add an element in the end of list.

> concatenate: **a+b**, join two or more lists

> append is same as std::vector, pre-allocate space, fast; concat is slow.

> cannot delete iteratively, since the **next()** will be also deleted

**List Comprehension:** list = [i for i in range(1,101)]

**Generator Expression:**

> list\_gen = (i for i in range(1,101))

> returns an iterator, generate element in demand.

> requires less memory

**Tuple:**

> Immutable, cannot be modified, static array.

> Can contain one or more types in a list, defined using ()

> With only one element: **tuple1 = (3,)**

> Tuple and list are **both indexed and iterable**.

> List usually stores a large collection of data with the **same type**

> Tuple usually stores a small collections of items with **various types**

**list():** Change tuple to list. **tuple():** change list to tuple

**Set:**

> unordered, mutable, no duplicate elements

> only len(a), min(a), max(a), x in a

>>> setA = {1,2,3,4}  
>>> setB = {3,4,5,6}  
>>> setA | setB #Union  
{1,2,3,4,5,6}  
>>> setA & setB #Intersection  
{3,4}  
>>> setA - setB #A-B  
{1,2}  
>>> setA ^ setB #(A|B)-A&B  
{1,2,5,6}

> **add(),** add single element. **update()**, add multiple elements.

> **delete()**, **discard()** remove element, delete will throw error if element is not exist.

> **pop()** delete and return an element

> **clear()** delete all elements

**Dictionary:**

> search key in the dict dict.get("apples") or dict["apples"], each key has a correspondent value

> each pair has a key(left) and a value(right)

> can store any type

> delete: dict.pop("apples") or del dict["apples"]

> clear(): clear all. copy(): make a copy. keys():return all keys. values(): return all values. items(): return all keys + values

**set(), list()** for list and set

**Hashability:**

> Object’s id will not change until it lifetime ends

> List, set and dict cannot be hash.

**Access the global variable:**

> In a function, global variable with global can: modify, both mutable and immutable; read

> Global variable without global can: modify mutable only by append or sort, etc.; read

**Pass by assignment**: similar with pointer pass. When a mutable is passed, it will modify the original. For immutable variable, it will create a new object

**Lambda:**

>>> (lambda x:x)(10) # Identity function  
10  
>>> (lambda x: 'abc')(5) # Constant function  
‘abc'  
>>> (lambda x,y,z: x+y+z)(4,5,9) # Multiple arguments  
18

**Lambda in functions:**

>>> def func\_a(n)  
 return lambda x:x+n  
>>> f1 = func\_a(10)  
>>> f1(1)  
11  
>>> f1(2)  
12

**Variable store a function:**

> say\_hello = greet(), store the output.

> say\_hello = greet, store the function

> Their id are identical

> function can store in list, tuple, set, dict

> function can be passed as argument to functions

> function can be returned from function

**Closures:**

> remember the state and the environment

> returns a inner function

> preserve function state across function calls

**Decorators:**

> all decorators are closures

> for decorators, the outer function accepts a function as input arg

图片包含 图示

描述已自动生成

图示

描述已自动生成

> **int('-12.210')** throws an error, cannot be string

> **['a','b','c','d'][::-1]**即倒序排列，即**`['d','c','b','a']`**

> **['a','b','c','d'][-1] = ['d']**

> **['a','b','c','d'][1:-1] = ['b','c']**

> **[].append('IT5001')** 不打印任何结果，应该选none

> **3 in {1,2,{3,4}}** 此set包含了一个set，而set中的元素必须是可散列的(hashable)，然而set本身是不可散列的，所以此set是非法的

> **(lambda x: x (3))(lambda x: x\*4)**前面一个lambda是带常数的，并且没有附加计算，所以为3。第一个lambda的输出可以看作是第二个lambda的输入。即这个表达式可以写为lambda x: x\*4 (3)。所以输出为12

> **1(2+3)%4**此表达式中的1会被python理解为函数名称，所以会抛出错误

> **[1, 2] + (3, 4)** list和tuple不能相加，因为是不同的数据结构，可以将list或tuple转换成对方的结构后相加。

> **x = [5, 0, 0, 1] += 'IT'** string是一个可以被迭代的，所以IT会被拆分I和T，输出为**[5, 0, 0, 1, 'I', 'T']**

> **[1, 2, 3][4:5] and 'IT5001!'** 其中，**[1, 2, 3][4:5]**会返回一个空列表`[]`，and会返回第一个逻辑为false的值，如果所有制都为True，则返回最后一个值。空列表被看作是False，而任何非空的对象都会是True。所以此表达式会返回空列表`[]`

> **[[1, 2], [3, 4]][1][0]**第二个元素的第一个子元素，返回3\

> **list(filter(bool, [0, 1, 2]))** bool会测试每个元素的布尔值，然后filter会只保留布尔值为True的值，所以输出为[1,2]

> We say ‘x contains itself’ to mean that `x in x` gives True. As we have seen, a list ls can contain itself, such as when we do `ls.append(ls)`. However, a set/tuple can never contain itself.

def zero\_one(ls):

if len(ls) <= 1: return ls

if ls[0] and not ls[-1]:

return (ls[-1:] + zero\_one(ls[1:-1])+ ls[:1])

if ls[0]:

return zero\_one(ls[:-1]) + ls[-1:]

return ls[:1] + zero\_one(ls[1:])

print(zero\_one([1, 1, 0, 1, 0]))

将0排列在1前面

def f29(ls):

d = {}

res = set()

m = -1

for i in ls:

d[i] = d.get(i, 0) + 1

for k, v in d.items():

if v > m:

res = set()

m = v

if v == m:

res.add(k)

return res

找出输入的字符串中出现次数最多的字符

def f(s, c):

if not s:

return 0

rec = f(s[1:], c)

if s[0] == c:

return 1 + rec

return rec

print(f('IT5001 is the best!', 'i'))

找出输入字符串中包含特定字符的次数

def c(f, g):

return lambda x: f(g(x))

def f(x):

return x + 1

def g(x):

return x \* 2

print(c(f, g)(3))

等效与lambda x: x\*2+1 (3) = 3\*2+1 = 7

def f(g, x):

if isinstance(x, int):

return [g(x)]

result = []

for i in x:

result.extend(f(g, i))

return result

print(f(lambda x: x \* x, [[[1, [2]], [3], 4], 5, [6]]))

将所有的intger平方

文本

中度可信度描述已自动生成