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 el ement 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:

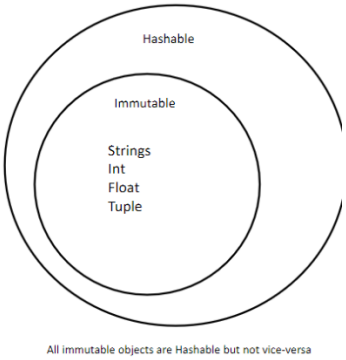
```
>>> (lambda x:x)(10) # Identity function
10
>>> (lambda x: 'abc')(5) # Constant function
‘abc'
>>> (lambda x,y,z: x+y+z)(4,5,9) # Multiple arguments
18
```

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

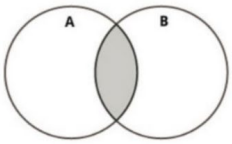
Data Type	Immutable	Hashable
Integer	Yes	Yes
Float	Yes	Yes
String	Yes	Yes
Tuple	Yes	Yes
List	No	No
Set	No	No
Dictionary	No	No



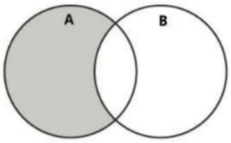
All immutable objects are Hashable but not vice-versa

Set Operations

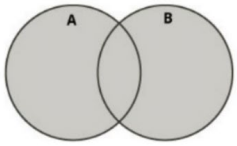
- Intersection



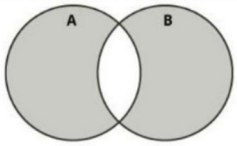
- A – B



- Union



- Symmetric Difference



> **`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

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Ch
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DE