immutables: 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)
In Python, 0 is falsy (evaluates to False), and ANY NON-ZERO number is truthy (evaluates to True). An existing function is TRUE.

Arithmetic

Arithmetic Precedence
Python follows the PEMDAS rule (Parentheses, Exponents, Multiplication and Division, Addition and Subtraction),

Operators	Associativity
() Highest precedence	Left - Right
**	Right - Left
+x , -x, ~x	Left - Right
*, /, //, %	Left - Right
+, -	Left - Right
<<, >>	Left - Right
&	Left - Right
^	Left - Right
|	Left - Right
Is, is not, in, not in, <, <=, >, >=, ==, !=	Left - Right
Not x	Left - Right
And	Left - Right
Or	Left - Right
If else	Left - Right
Lambda	Left - Right
=, +=, -=, *=, /= Lowest Precedence	Right - Left

Sorted(): converts the string to be sorted into a list of chars and then sorts it. E.g. Sorted('abc') -> ['a','b','c']
Print function can directly print integers and floats.

Important Notice: （不看丢分）
1. **'And' ranks higher than 'or' in precedence!!!**
2. **** has a right-to-left associativity!!!**

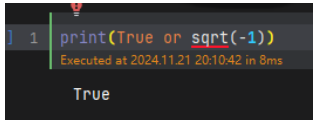
- Type(4/2) returns **a float.**

¡¡¡Note: In Python 3, the / operator always returns a float, even if the division result is an integer!!!

- // Integer (floor) division: Divides the first operand by the second and rounds to the nearest integer
  Type(4//2) returns **an integer.** (since it has been rounded)
- Print(m=n) causes an error (invalid keyword argument)
- **Int() can convert integer-like strings ('42'), floats (3.7), Boolean values (True), binaries (b'1010',2) into integers, but not others (like non-numeric strings 'aaa', '-12.210', complex numbers etc**
- Int(float number) **always gives the truncated /floored version.** (e.g. int(13/4) = 3, int(-13/4) = -3, int(9.0) = 9)
- Round() function:
  - If the number is **not halfway** (e.g., 3.2, 4.7), it is rounded to the **nearest** integer as usual.
  - If the number is **exactly halfway** between two integers (e.g., 2.5, 3.5, 4.5), it is rounded to the **nearest even integer**. (Bankers' Rounding)
  - You can specify how many digits you'd like to round by passin' as a param
- In Python, **boolean values True and False are treated as integers in arithmetic operations**:
  True is equivalent to 1, False is equivalent to 0
  So, the expression 1 != 2 (which is True) becomes 1.
- The sum() function in Python is used to calculate **the sum of all the elements in an iterable** (such as a list, tuple, or set). It adds up all the values in the iterable and returns the total.
- The OR operator returns the first truthy value (Short circuit logic)or the last falsy value.
- print('0' * 1 + 2 * '3') This line prints 033. * is commutative.
- Infinite recursions raise **recursion error.**
- Int can't concatenate directly with string. '4' + 0 returns error.
- **False == True == False is equivalent to (False == True) and (True == False).** This is called chained comparisons.
- The expression 11 & 4 performs a **bitwise AND operation** on the integers 11 and 4. 0b1011 AND 0b0100 is 0

Data Types

- Given x = 1, -x is **its opposite number**.
- True is 1 and can be compared with numbers. (true < 5001 returns True)
- Equality comparison: either use **==/!=** (for values) or **is/is not** (for addresses) but **never a mixture of both.** (not == gives a syntax error)
- You **can** use **multiple comparisons** in a chain in python.
  print('c' < 'c++'<'java') returns true.
- You can even write like this: print(**0 < 2 > 1**), which is also true.
- Math.floor() returns the floored value of a float, which is **an int.**
- Packages are of type **module**.
- You **can't write sqrt(-1) directly in a fresh idle without importing math.** Even if you imported math and wrote math.sqrt(-1), it still raises domainError. Thus, if you see this in MCQ, directly choose Error.
- **** But be careful of the following exception: Short Circuit logic!!!****
- **Input() returns whatever the user inputs as a string.**



Strings

String concatenation: +
The first index of a string is 0.
String slicing
Syntax: s[start: stop: step]
Start is **included** and stop is **excluded.**
By default, the start position is 0, stop position is the last index and step is 1.

- String slicing:
  [start:end:step]
- The **start position** in string slicing **can exceed the maximum index** (which defaults to the character of **the last index**) and no error occurs. E.g. 'abcde'[6:7] gives ''
- The **end position** in string slicing **can also exceed the maximum index value** and no error will occur. E.g. 'abcde'[-1:7] gives 'e'
- **However, if you access an element whose index is out of range, it will trigger an error (index out of bound)**
- A slice [start:end] where **start > end** always **results in an empty string.**
- '' is a substring of every string.
- Int(string) only accepts strings that represent **integer** values.
- Strings can only concatenate str (not "NoneType") to str
- **String comparison**
  Char-wise, compares ASCII values. The upper case letters have **smaller** ASCII values than lower case letters.

File IO

模式	描述
r	只读模式，文件必须存在，默认模式。
w	写入模式，文件不存在时创建新文件，存在时清空内容。
a	追加模式，从文件末尾开始写入，文件不存在时创建新文件。
b	二进制模式（与其他模式组合），用于处理非文本文件（如图片、音频等）。
t	文本模式（与其他模式组合），用于处理文本文件（默认）。
+	读写模式（与其他模式组合），允许同时读取和写入文件。

模式	描述
rb	以二进制模式读取文件。
wt	以文本模式写入文件。
w+	清空文件并以读写模式打开。
a+	以追加模式打开文件，并允许读取。

String and List conversion

1. Str -> List

- List(str) cuts **every single char** into a separate element in the list.
  list('123') -> ['1', '2', '3'] (Notice the difference with string.split()!)
  s = 'KDnuggets is a fantastic resource'
  ['K', 'D', 'n', 'u', 'g', 'g', 'e', 't', 's', ' ', 'i', 's', ' ', 'a', ' ', 'f', 'a', 'n', 't', 'a', 's', 't', 'i', 'c', ' ', 'r', 'e', 's', 'o', 'u', 'r', 'c', 'e']
- **Split strings into lists of smaller substrings (word-wise)** with split().
  s = 'KDnuggets is a fantastic resource'
  >>> print(s.**split**()) ['KDnuggets', 'is', 'a', 'fantastic', 'resource']

2. List -> Str

- str([1,2,3]) -> '[1, 2, 3]'
- Join list element strings into single string in Python using **join().**

Lists

- [1] * 2 = [1,1]
- range(1, 10, 2):
- range() generates a sequence of numbers starting at 1 (the first argument) and ending before 10 (the second argument), with a step of 2 (the third argument).
- The **end** position in list slicing **can exceed the maximum index value** and no error will occur.
- **However, if you access an element whose index is out of range, it will trigger an error (index out of bound)**
- The line print([].append('IT5001')) returns **None** because the append() method in Python modifies the list in place and **does not return the modified list itself**.
- print([] + [1] * 2) returns [1,1] instead of [2]. It doesn't multiply the numeric values but duplicates the lists
- **¡¡¡ The difference between append, extend and += !!! (High Frequency)**
  - List.**append**(element) appends the element **as one element**.
  - List.**extend**(element) or += appends the **individual element of the element** if it is iterable.
  - Using += is equivalent to calling extend().
    For example, x = [5,0,0,1]
    x.append('IT') => [5,0,0,1,'IT']
    x += ('IT') => [5,0,0,1,'I','T']
    x.extend('IT') => [5,0,0,1,'I','T']
    - Start from innermost layer when simplifying complicated expressions of list, evaluate from left to right for outer layers
    - You can't write [...][[1]]. TypeError: list indices must be integers or slices, not list
    - The for loop iterator is created at the start and **does not adjust to changes in the list caused by l.pop(0)**. It's possible that **the list may change** while iterating.

For loops with range
a) forward
```
for i in range(5):
    print(i)
```
output: 0, 1, 2, 3, 4
b) backward
```
for i in range(5, -2, -1):
    print(i)
```
output: 5,4,3,2,1,0,-1

List Comprehension

Dealing with super-long & complicated list comprehensions
For list comprehension in this structure
[expression **for** outer_var **in** outer_iterable **for** inner_var **in** inner_iterable **if** condition]
It can be expanded into the normal form of double for loop as follows:

```
result = []
for outer_var in outer_iterable:    # Outer loop
    for inner_var in inner_iterable:    # Inner loop
        if condition:
            result.append(expression)
```

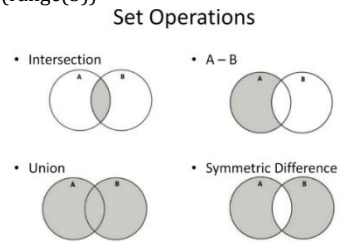
Tuples

- **(i + 1 for i in range(3)) IS NOT A TUPLE!!! It is a generator object Which can be iterated over**

- Tuples are immutable objects. **You can not append elements via append() to a tuple. But you actually can write (1,2,3) + (4,)**
- 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.**
- **Differentiate an int from a single-element tuple**
  This is an int: (3); This is a single-element tuple: (3,)
- **Tuples don't have extend() method. It's for lists only.**
- [1,2] + (3,4) definitely can't be executed and raises an error but **[1,2] + ([3,4]) can work** since ([3,4]) is just a list in parentheses
- **Never write tp += I where tp is a tuple and I is an integer (+= (I,))**
- Repeated concatenation for every iteration **has complexity of O(n^2)**
  A better alternative in O(n): tp = tuple(range(5))

Sets

- Add an element to the set: add()
  Remove the specified item:
  discard() -- no ERROR if not found, no default return
- | Union & Intersection – Exclude ^ Symmetric Difference



Dictionaries

- The **dict()** constructor is used to create a dictionary.
  It expects either:
  - An iterable where each element is a **key-value pair** (like a list of tuples or another dictionary), or
  - Keyword arguments that will become key-value pairs in the dictionary.

  Correct usage: **dict([('key1', 'value1'), ('key2', 'value2')])**
- **Dictionaries cannot be added to sets** in Python. This is because dictionaries are **unhashable**, and **only hashable objects** can be added to sets.
- In dictionaries, duplicate keys **will be overwritten by the one that appears the last.**
- Delete a key: del dict[key]
- Dictionary values are mutable **but not for keys.**
- print(list({1:2, 3:4})) returns [1,3].
- If a dictionary has duplicate keys, **the last corresponding value overwrites the previous ones.**
- **You can't change a dictionary's size during iteration!!!**
- **Retrieving a key's value: dictionary.get(key, default)**
  If the key you are looking for **doesn't exist, return None**

.items(): Returns all keys + values

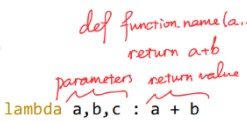
Map & Filter

- The map() function **executes a specified function for each item in an iterable.** The item is sent to the function as a parameter.
- The filter() function **returns an iterator where the items are filtered through a function** to test if the item is accepted (Truthy) or not (Falsy).
- **** for filter(), if you don't find an obvious Boolean function, think of when the function expression might return a TRUTHY value!!! ****
- L = [**9,2,1,3,4,5,6**]
- print(list(map(lambda x: x>**2,** L)))
  [True, False, False, True, True, True
  , True]
- print(list(filter(lambda x: x>**2,** L)))
  [9, 3, 4, 5, 6]
  a = map(lambda x: x // 2, [1, 2, 3])
  print(max(a)) -> 1
  print(min(a)) -> ValueError: min() iterable argument is empty
- **Reduce(function, params):** applies a binary function (a function taking two arguments, max in this case) **cumulatively** to the elements of the iterable, **reducing it to a single value.**

BE CAREFUL OF THE FOLLOWING DIFFERENCES!
x = tuple(map(lambda x:x%3, [2,4,6])) -> (2,1,0)
x = tuple(map(lambda x:x%3 != 0, [2,4,6])) -> (True, True, False)
Map and filter are both disposable. The contents are cleared after their 1st usage, after which they'll be depleted.

Lambda

- In Python, lambda expressions do not require at least one positional argument. **A lambda function with no arguments is perfectly valid.** (like lambda:x). In this case, **upon a call (lambda: print('1'))(), the code on the RHS will execute.** (like when you call a function without parameters)



- If we do not define functions using def and we only define functions using lambda expressions, we cannot write recursive functions (X)
  Lambda can be recursive.
- [lamda x: x + i for i in range(3)] >> **[x + 2, x + 2, x + 2]**

Copying

!!! copy() is a shallow copy that only copies the outermost layer and stores them into a new address!!! A change in inner layer still affects both

1. **Primitive data copying**
   For primitive data copying, if we assign the variable whose previous value was copied a new value, then **the variable that copies values isn't affected.**
2. **List copying**
   For list copying, if we modify a value in the original list, then **the list that copies that is also affected by the change (since they point to the same list object).** (See example below)

```
>>> x = 1
>>> y = x
>>> x = 2
>>> print(y)
>>> listx = [1,2,3]
>>> listy = listx
>>> listx[0] = 999
>>> print(listy)
[999, 2, 3]
```

2D arrays

Right and wrong ways to construct a matrix

Correct:

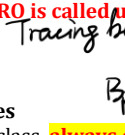
```
def create_matrix(r,c):
    row = [0] * c
    return [row.copy() for _ in range(r)]
```

Wrong: (since a change on single element will change every column)

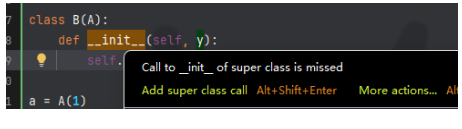
```
def create_matrix(r,c):
    row = [0] * c
    output = []
    for i in range(r):
        output.append(row)
    return output
```

OOP

1. **Modifying instance variables**
   - class Pokemon:
   - def __init__(self, name, power):
   - self.name = name
   - self.slogan = name + ' ' + power
   - slogan is **not dynamically linked** to name.
   - Once the slogan is set during initialization, **it remains independent of the name attribute.** If you want slogan to reflect changes to name, you need to explicitly update it or use a different implementation.
   - 就是说如果你改了名，需要用到你名字的地方还是用的原名
2. **Diamond inheritance**
   In Python, when a class inherits from multiple parent classes, **only the __init__ method of the first parent in the MRO is called unless explicitly overridden.**
   MRO Example (right hand side)
   Resulting MRO: [D, B, C, A, object].



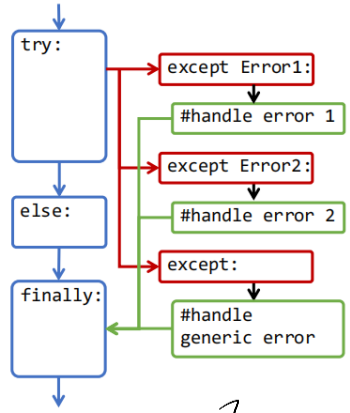
3. **Properly inheriting attributes in subclasses**
   When overriding the __init__ method in a subclass, **always call the parent class's __init__ method** (using super() or explicitly) if you want to inherit its initialization logic.



4. If your subclass **DOESN'T** have the init function, then **the parameters of the parent class are INHERITED by default.** Functions of a parent class are inherited by default unless overridden. The constructor would also be inherited though

Try except (finally)

See flowchart



Algorithm Complexities

- Linear search: O(n)
- Binary search: O(log n)

- Selection/Bubble/insertion sort: O(n^2)
- Merge sort: O(n log n)

Memoization:

a. Create a dictionary to remember the answer computed before.
b. **If the answer is later used, get it directly from the dictionary.**
c. Otherwise, compute the answer and put it into the dictionary (see example below)

Memoizing using a list is at least as efficient (if not slightly more efficient) than using a dictionary because of lower memory overhead and O(1) access.

```
fibans = {}

def fibm(n):
    if n in fibans.keys():
        return fibans[n]

    if n == 0:
        ans = 0
    elif n == 1:
        ans = 1
    else:
        ans = fibm(n-1) + fibm(n-2)
    fibans[n] = ans
    return ans
```

Some Useful Patterns

1. **Deep Squaring**

```
def deepSquare(seq):
    if seq == []:
        return seq
    elif type(seq) != list:
        return seq * seq
    else:
        return
[deepSquare(seq[0])] +
deepSquare(seq[1:])
```

2. **Deep Tuple**

```
def deep_tuple(s):
    if not isinstance(s, list): 3
return s
    return
tuple(deep_tuple(x) for x in s)
```

3. **Shallow Map**

```
def flatten(seq):
 if not seq:
    return seq
 elif type(seq) != list:
    return [seq]
 else:
    return flatten(seq[0]) +
flatten(seq[1:])
```

4. **Sum of all digits in one line**

```
def digitsum(n): return
sum(map(lambda x: int(x),
list(str(n))))
```

5. **Fibonacci Iterarive Version**

```
def fib_iterative(n):
    a, b = 1, 1
    for _ in range(2, n + 1):
        a, b = b, a + b
    return b
```

6. **Checking for Palindromes**
   phrase.find(phrase[::-1]) -> return 0 for palindromes, -1 else
7. **Finding all prime numbers <= n**
   set(range(1, n)) - set([j for i in range(2, int(n ** 0.5 + 1)) for j in range(i * 2, n, i)]
8. **Printing file list line by line:**
   [line.strip() for line in open(filename)]

9. **Binary Search**

```
While left <= right:
    mid = (left + right) // 2
    if ls[mid] == key: return True
    elif ls[mid] < key: left = mid +
1
    elif: ls[mid] > key: right = mid
- 1
```