Importance ■■■■■ most asked ■■■□□ often ■□□□□ rarely

Operators and precedence ■■■■■

Highest to lowest: `()` → `**` (right to left) → unary `-` → `*` `/` `//` `%` → `+` `-` → comparisons → `not` → `and` → `or`.

EXPRESSION	VALUE
`7 / 2`	`3.5`
`7 // 2`	`3`
`-7 // 2`	`-4`
`-7 % 3`	`2`
`2 ** 3 ** 2`	`512`
`3 * 2 ** 2`	`12`
`0 and 5`	`0`
`4 and 5`	`5`
`0 or 'A'`	`'A'`
`5 > 3 or 1 / 0`	`True`
`1 < 3 < 2`	`False`
`True + True`	`2`
`'5' * 3`	`'555'`
`int('7') + float('2.5')`	`9.5`

! `/` always gives a float. `//` rounds **down**, so `-7 // 2` is −4. `and`/`or` return one of the **operands**, not always True/False, and stop early: `1 / 0` above is never evaluated.

Strings ■■■■■

On `s = 'Computer Science'` (16 characters, indexes 0–15 or −16 to −1). `s[start:stop:step]` includes start, **excludes stop**.

SLICE	RESULT
`s[0:4]`	`'Comp'`
`s[-7:]`	`'Science'`
`s[::2]`	`'Cmue cec'`
`s[::-1]`	`'ecneicS retupmoC'`
`s[-3:0:-3]`	`'ncruo'`
`s[2:9:3]`	`'mt '`

METHOD	RESULT
`s.find('e')`	`6`
`s.find('z')`	`-1`
`s.index('S')`	`9`
`s.count('e')`	`3`
`s.split()`	`['Computer', 'Science']`
`s.split('e')`	`['Comput', 'r Sci', 'nc', '']`
`s.partition(' ')`	`('Computer', ' ', 'Science')`
`s.replace('e', '*')`	`'Comput*r Sci*nc*'`
`s.title()`	`'Computer Science'`
`'hello world'.capitalize()`	`'Hello world'`
`'-'.join(['a','b','c'])`	`'a-b-c'`
`'  hi  '.strip()`	`'hi'`
`'abc123'.isalnum()`	`True`
`'TariffWar'.find('war')`	`-1`

! Strings are **immutable**: `s[0] = 'K'` gives **TypeError**. Methods return a new string; `s` is unchanged unless you assign it back. `find()` gives **−1** if missing; `index()` raises ValueError.

Lists ■■■■■

Each line starts again from `L = [5, 2, 3, 4, 2]`.

CODE	L AFTERWARDS, OR THE OUTPUT
`L.append([6, 7])`	`[5, 2, 3, 4, 2, [6, 7]]`
`L.extend([6, 7])`	`[5, 2, 3, 4, 2, 6, 7]`
`L.insert(1, 9)`	`[5, 9, 2, 3, 4, 2]`
`L.remove(3)`	`[5, 2, 4, 2]` (first 3 only)
`print(L.pop());` `print(L)`	`2` `[5, 2, 3, 4]`
`print(L.pop(0));` `print(L)`	`5` `[2, 3, 4, 2]`
`L.index(4)`	`3`
`L.count(2)`	`2`
`L.sort(reverse=True)`	`[5, 4, 3, 2, 2]`
`sorted(L), L`	`[5, 4, 3, 2, 2] [5, 2, 3, 4, 2]` (sorted makes a new list)
`L[1:3] = [8]`	`[5, 8, 4, 2]`
`del L[1]`	`[5, 3, 4, 2]`
`max, min, sum, len`	`5 2 16 5`
`M = L; M.append(0); L`	`[5, 2, 3, 4, 2, 0]` **alias**
`M = L[:];` `M.append(0); L`	`[5, 2, 3, 4, 2]` copy

! `append` adds **one** item (a list stays a list inside); `extend` adds each item. `sort()` changes L and returns **None**; `sorted()` returns a new list. `M = L` is the same list under two names.

Dictionaries ■■■■■

```
drs = {'Kerala': 3, 'Goa': 4, 'Kerala': 1}
print(drs)
```

```
{'Kerala': 1, 'Goa': 4}
```

```
d = {'name': 'Mahi', 'age': 23}
print(d.get('age', 'Not Found'), d.get('city', 'Not Found'))
```

```
23 Not Found
```

```
test = {1: 'A', 2: 'B', 3: 'C'}
del test[1]
test[1] = 'D'
del test[2]
print(len(test), test)
```

```
2 {3: 'C', 1: 'D'}
```

```
d = {'a': 1}
d.update({'a': 5, 'b': 2})
print(d)
print(d.pop('a'), d)
```

```
{'a': 5, 'b': 2}
5 {'b': 2}
```

`for k in d`: gives the **keys**; use `d[k]` for the value. `d.keys()`, `d.values()`, `d.items()` give keys, values, (key, value) pairs.

! Keys are unique: a repeated key keeps its first place but takes the **last** value. `d['city']` for a missing key raises **KeyError**; `get()` doesn't.

Conditions and loops

RANGE	GIVES
`range(5)`	0 1 2 3 4
`range(2, 8, 2)`	2 4 6 (stop excluded)
`range(10, 0, -3)`	10 7 4 1
`range(5, 2)`	nothing: start already past stop

```
for i in range(1, 10):
    if i % 4 == 0:
        break
    if i % 2 == 0:
        continue
    print(i, end=' ')
```

```
1 3
```

```
n = 25
c = 0
while n > 0:
    n //= 3
    c += 1
print(n, c)
```

```
0 3
```

`break` leaves the loop; `continue` skips to the next round. `print(x, end='-')` ends with - instead of a new line; `sep=''` changes the gap between values.

Tuples

```
t = (5)
u = (5,)
print(type(t), type(u))
```

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

```
tuple1 = (1, 2, 3)
tuple2 = tuple1
tuple1 += (4,)
print(tuple1 == tuple2, tuple1, tuple2)
```

```
False (1, 2, 3, 4) (1, 2, 3)
```

```
tup1 = (10, 20, 30, 40, 50, 60, 70, 80, 90)
print(tup1[-2:-5])
print(tup1[-5:-2])
```

```
()
(50, 60, 70)
```

! A one-item tuple needs a **comma**. Tuples are immutable: `t[0] = 9` gives TypeError, and += makes a **new** tuple. A slice with a positive step and start after stop is **empty**.

Data types, mutability, tokens

	MUTABLE (CAN CHANGE)	IMMUTABLE
Types	list, dictionary	int, float, complex, bool, string, tuple

Tokens: keywords (`if`, `for`, `True`, `None`), identifiers (names), literals (`5`, `'hi'`), operators, punctuators. An **identifier** starts with a letter or _, has only letters, digits, _, and is not a keyword: `_total1` is valid; `1total`, `my-name`, `for` are not.

Write a statement (built-in methods only)

TASK	STATEMENT
string S to uppercase	`S.upper()`
number of elements in list L	`len(L)`
L in descending order, into L1	`L1 = sorted(L, reverse=True)`
insert 2500 at the second position	`L.insert(1, 2500)`
index of first "error" in log	`log.find('error')`
remove leading * from tops	`tops.lstrip('*')`
change "World" to "$" in Msg	`Msg = Msg.replace('World', '$')`
first three elements of L	`L[:3]`

Find the error (2 marks)

Rewrite the code and underline each correction. These are the slips papers plant most.

WRONG	RIGHT
`Def f():` · `define f():`	`def f():`
`30 = num`	`num = 30`
`for k range(0, 5):`	`for k in range(0, 5):`
`range[0, 7]`	`range(0, 7)`
`if x == 3` (no colon)	`if x == 3:`
`if x = 3:`	`if x == 3:`
`Elseif` · `else if`	`elif`
`print('Total: ' + 5)`	`'Total: ' + str(5)`
`Print('hi')`	`print('hi')`
body not indented under `if`	indent it

What's the output?

Programs in the style of 2 and 3-mark questions. Cover the right column and answer; every result was run.

CODE	OUTPUT

`s = 'Exam2026'`
`m = ''`
`for i in range(len(s)):`
`  if s[i].isupper():`
`  m = m + s[i].lower()`
`  elif s[i].islower():`
`  m = m + s[i].upper()`
`  else:`
`  m = m + '#'`
`print(m)`
	eXAM####

`L = [81, 82, 18, 49]`
`for i in range(len(L)):`
`  if L[i] % 3 == 0:`
`  L[i] //= 3`
`  else:`
`  L[i] += 1`
`print(L)`
	[27, 83, 6, 50]

`count = {}`
`for ch in 'banana':`
`  count[ch] = count.get(ch, 0) + 1`
`print(count)`
	{'b': 1, 'a': 3, 'n': 2}