

General Reminder

<code>r.randint(x,y)</code>	Generate a random integer between x and y (import random as r).
<code>input(msg)</code>	Prompt the user with msg and take the user input.
<code>type(var)</code>	Returns the type of var.
<code>type(n, b, d,)</code>	Dynamically create a class named n . This class inherits all classes in b (tuple). d is a dictionary containing attributes and member method.
<code>id(var)</code>	Returns the memory address of var .
<code>dir(var)</code>	Returns a list of all attributes and methods of var .
<code>int(var)</code>	Convert var to a integer.
<code>float(var)</code>	Convert var to a float.
<code>str(var)</code>	Convert var to a string.
<code>len(var)</code>	Returns the length of a string or a list.
<code>pass</code>	Used to keep an indentation empty avoiding IndentationError .
<code>.copy()</code>	Creates a new object but nested objects still reference the original.
<code>.deepcopy()</code>	Creates a new object with completely new copies of all nested objects.
<code>del var</code>	Deletes var from memory.

Operators

Symbol	Name	Type
+	addition	Arithmetic
-	subtraction	Arithmetic
*	multiplication	Arithmetic
/	division	Arithmetic
%	modulo	Arithmetic
**	power	Arithmetic
//	div	Arithmetic
and	logical and	Boolean
or	logical or	Boolean
not	logical not	Boolean
in	in	Membership
==	equal	Comparison
!=	not equal	Comparison
>	greater than	Comparison
<	less than	Comparison
>=	greater than or equal	Comparison
<=	less than or equal	Comparison

Error Handling

```
try:
    # risky operation
except ex:
    # runs if an exception of type ex is raised
else:
    # runs if no exception is raised
finally:
    # Runs regardless of what happens
```

Data Structures

list	[e_1, ...]	ordered, changeable, duplicates.
tuple	(e_1, ...)	ordered, unchangeable, duplicates.
set	{e_1, ...}	unordered, unchangeable, no duplicates, unindexed.
dictionary	{a_1:b_1, ...}	ordered, changeable, no duplicates.

Typing

import typing to use type hints.

Sequence	ordered container supporting indexing and slicing.
List	mutable ordered sequence, supports indexing and append.
Tuple	immutable ordered sequence, fixed-size.
Set	unordered collection of unique elements.
Dict	mapping of keys to values.
Mapping	abstract read-only mapping interface.
MutableMapping	mapping that supports mutation.
Iterable	can be iterated with for-loops.
Iterator	yields items on demand.
Sized	supports len().
Hashable	can be used as dict key or set element.
Optional[T]	either T or None.
Union[A,B]	value may be of type A or B.
Any	accepts any type.

sets Methods

<code>.add(e)</code>	Add e in the set.
<code>.update(lst)</code>	Add all elements from lst in the set.
<code>.remove(e)</code>	Remove e in the set.
<code>.union(lst)</code>	add all elements from lst
<code>.intersection(lst)</code>	keep only the elements that are both in its
<code>.difference(lst)</code>	remove all element of lst in the set
<code>.symmetric_difference(lst)</code>	

lists Methods

```
lst1 = [e_1, e_2, e_3] # [e_1, e_2, e_3]
lst2 = 5 * [a] # [a, a, a, a, a]
lst3 = [a for i in range(3)] # [a, a, a]
```

<code>list(var)</code>	Convert a set or tuple to a list.
<code>lst[i]</code>	Access the ith element in the list.
<code>.append(a)</code>	Adds a to the end of the list.
<code>.insert(i,e)</code>	Insert element e at index i .
<code>.pop()</code>	Remove the last element, return the removed value.
<code>.pop(i)</code>	Remove element at index i , return the removed value.
<code>range(n)</code>	Create a list with all integers from 0 to n .

dictionary Methods

```
dic = {"max":22, "ugo":40, "cyp":21}
dic["max"] # -> 22
```

Regular Expressions(REGEX)

Functions

Let **e** be a regular expression and **s** and **s2** be strings.

```
import re
```

include python REGEX library.
.findall(e, s) Returns a list containing all matches of **e** in **s**.
.search(e, s) Returns a Match object if there is a match of **e** anywhere in **s**.
.split(e, s) Returns a list where **s** has been split at each match of **e**.
.sub(e, s2, s) Replaces one or many matches of **e** with **s2** in **s**, optional parameter: number of replacements.
Match Object A Match Object is an object containing information about the search and the result. Can act as a boolean that is true if not empty.

REGEX Syntax

Metacharacters:

[]	A set of chars	*	Zero or more occurrences
\	special sequence and escape special char	+	One or more occurrences
.	Any character except \n	?	Zero or one occurrences
^	Starts with		Specify number of occurrences
\$	Ends with	()	Capture and group
	Either or		

Set Syntax by example:

• **[aBc]** matches a single character: a, B, or c. • **[^aBc]** matches any single character except a, B, or c. • **[a-z]** matches any lowercase letter a through z. • **[a-zA-Z]** matches any letter (uppercase or lowercase). • **[0-9]** matches two consecutive digits (00-99).

Special Sequences:

\A	Match at start of string	\d	Any digit [0-9]
\b	Word boundary	\D	Any non-digit
\B	Not a word boundary	\s	Any whitespace
\Z	Match at end of string	\S	Any non-whitespace
\w	Any word character	\W	Any non-word character

REGEX Backreferences

Refer to earlier capturing groups in the pattern or in replacements. Prefer raw strings (e.g., **r"..."**) to avoid double escaping.

\1, \2, ...	Backreference to the nth capturing group in the pattern.
(?P<name>...)	Named capturing group.
(?P=name)	Backreference to the named group in the pattern.
\1, \2	In replacement: refer to groups 1, 2 (works in re.sub()).
\g<name>	In replacement: recommended form for numeric/named groups.

File Handling

open(s, arg)	returns a file object from a file at path s . file object can be iterated line by line.
.read()	returns a string version of a file.
.read(n)	returns a string version of a n first characters in file.
.readline()	returns return the next line of the file as a string.
.close()	Manually close a file.
.write(s)	writes string s to the file.
.writelines(s)	writes a list of strings s to the file.

arguments for open(): • **r:** read only (default). • **w:** write only, creates or overwrites a file. • **a:** append only, creates file if not existing. • **b:** binary mode. • **t:** text mode (default). • **x:** create, fails if file exists. • **+:** read and write.

with statement

The **with** statement manages resources resource management by automatically handling setup and cleanup tasks. For file handling, it ensures that files are properly closed when the block exits, even if an exception occurs.

```
with open("myFile.txt") as f:
    myString = f.read()
```

Object Oriented Programming(OOP)

```
class ClassName(parent1, parent2, ...):
    var = value # class attribute with default value
    def __init__(self, arg1, ...): # constructor
        self.attr1 = arg1
    ...
    def method1(self, ...):
        # code
```

self	reference to the current instance of the class.
__init__	constructor method called when an object is created.
__del__	destructor method called when an object is deleted.

Operator Overloading

__str__(self)	Overload str() .
__repr__(self)	Overload repr() (used by print()).
__add__(self, o)	+ addition
__sub__(self, o)	- subtraction
__mul__(self, o)	* multiplication
__truediv__(self, o)	/ division
__floordiv__(self, o)	// floor division
__mod__(self, o)	% modulo
__pow__(self, o)	** power
__neg__(self)	unary -
__pos__(self)	unary +
__abs__(self)	abs()
__eq__(self, o)	== equality
__ne__(self, o)	!= inequality
__lt__(self, o)	< less than
__le__(self, o)	<= less or equal
__gt__(self, o)	> greater than
__ge__(self, o)	>= greater or equal

Strings

str(var)	Convert var to a string.
.lower()	Returns the string in lowercase.
.upper()	Returns the string in uppercase.
.strip()	Removes whitespace from the beginning and end of the string.
.join(lst)	Joins all elements in lst into a single string, separated by the original string.
r"..."	Raw string, ignores escape sequences.
f"...{var}..."	Formatted string literal, inserts var inside the string.
s[i:j]	Slicing, returns substring from index i to j-1 .

Basic Syntax

```
if condition_1:
    # code if condition_1 is true
elif condition_2:
    # code if condition_2 is true and condition_1
    # is false
else:
    # code

def function(a:type1, b:type2 = value, ...) -> rType:
    # code
    return # value of type rType

for i in lst:
    # for each element in lst

while condition:
    # runs while condition is true
```

Performance Tips

Numba

Numba is a Just-in-Time (JIT) compiler that translates a subset of Python and NumPy code into fast machine code at runtime.

```
from numba import jit
@jit()
def my_function(...):
```

code

jit settings:

- nopython=True** Disable the use of the Python interpreter within the compiled function for maximum performance.
- forceobj=True** Force the use of object mode, allowing compilation of a wider range of Python constructs at the cost of performance.
- looplift=True** Enable loop lifting optimizations to improve performance of loops.
- parallel=True** Enable automatic parallelization of loops within the function.
- cache=True** Cache the compiled machine code to disk for reuse in future runs.

Data Structures Performance Comparison

	Element access	Membership checks	Middle insert/delete	Append if reserved
Name				
list	$O(1)$	$O(n)$	$O(n)$	$O(1)$
tuple	$O(1)$	$O(n)$	$O(n)$	N/A
dictionary	$O(1)$	$O(1)$	$O(n)$	$O(1)$
set	N/A	$O(1)$	N/A	$O(1)$