Python Intermediate

What topics are we going to look into?

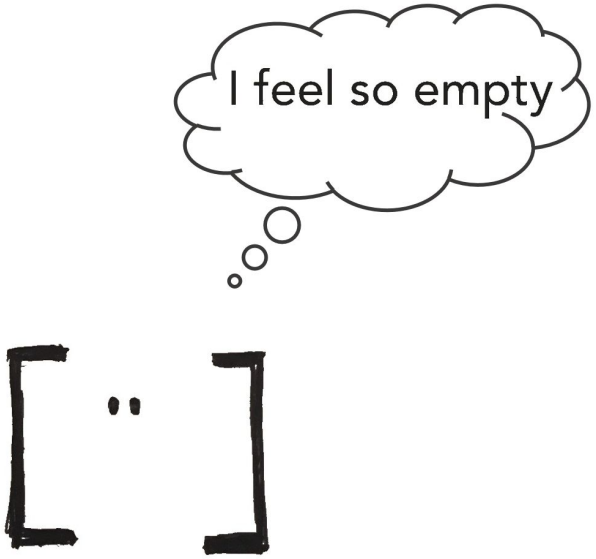
- Lists
- Tuples
- Sets
- Dictionaries
- Exception Handling
- Recursion

1st Topic

List

Lists

Lists are dynamic sized arrays. A single list can contain datatypes like Integers, Strings, Objects.



Creating a List

```
# List of Integers
list1 = [1, 2, 3]
print(list1)

# List of Integers, Floats, and Strings
list2 = [6.9, "Python", 420, "is", 0.21, "hard"]
print(list2)
```

```
[1, 2, 3]
[6.9, 'Python', 420, 'is', 0.21, 'hard']
```

```
list = [2] * 3
print(list)
```

```
[2, 2, 2]
```

The size of the list

```
# Size of empty list
list1 = []
print(len(list1))

# Size of List with Integers, Floats, and Strings
list2 = [6.9, "Python", 420, "is", 0.21, "hard"]
print(len(list2))
```

```
0
6
```

Adding element to a list

```python
# Append
list = []
print("Starting List: ")
print(list)
list.append(1) # [1]
list.append(2) # [1, 2]
list.append(3) # [1, 2, 3]
print("After Append: ")
print(list)

list = []
print("Cleared List: ")
print(list)
for i in range (4):
    list.append(i)
print("Append with for loop: ")
print(list)

list2 = ["UwU", "OwO"]
list.append(list2)
print("List after append list2: ")
print(list)
print("List2: ")
print(list2)
```

```
Starting List:
[]
After Append:
[1, 2, 3]
Cleared List:
[]
Append with for loop:
[0, 1, 2, 3]
List after append list2:
[0, 1, 2, 3, ['UwU', 'OwO']]
List2:
['UwU', 'OwO']
```

```python
# Insert
list = [1, 2, 3]
print("Initial List: ")
print(list)

list.insert(1, 10) # [1, 10, 2, 3]
list.insert(0, 99) # [99, 1, 10, 2, 3]
print("After Insert: ")
print(list)
```

```
Initial List:
[1, 2, 3]
After Insert:
[99, 1, 10, 2, 3]
```

```python
# Extend
list = [1, 2, 3]
list.extend([4, 5, 6])
print(list)

list = [1, 2, 3]
list.append([4, 5, 6]) # Not the same
print(list)
```

```
[1, 2, 3, 4, 5, 6]
[1, 2, 3, [4, 5, 6]]
```

Access to an element in the list

```
# Accessing Elements
list = [1, 2, 3]
print("Element at index 0: ")
print(list[0])
print("Element at index 2: ")
print(list[2])

list = [[1, 2], [3], 4]
print(list[0][0]) # 1
print(list[0][1]) # 2
print(list[1]) # [3]
print(list[1][0]) # 3
print(list[2]) # 4
```

```
Element at index 0:
1
Element at index 2:
3
1
2
[3]
3
4
```

Negative Indexing

```
# Negative Indexing
list = ["I", "am", "a", "list"]
print(list[-1])
print(list[len(list) - 1])

print(list[-2])
```

```
list
list
a
```

Removing element from a list

```
# Remove
list = ["A", "B", "C", "D", "E"]
list.remove("A")
print(list)
```

```
['B', 'C', 'D', 'E']
```

```
# Pop
list = ["A", "B", "C", "D", "E"]
list.pop(0)
print(list)
```

```
['B', 'C', 'D', 'E']
```

Slicing a list

```
# Slicing
list = ["A", "B", "C", "D", "E", "F", "G", "H"]
sliced = list[1:5]
print(sliced)

sliced = list[2:]
print(sliced)

sliced = list[:-3]
print(sliced)

sliced = list[::-1]
print(sliced)
```

```
['B', 'C', 'D', 'E']
['C', 'D', 'E', 'F', 'G', 'H']
['A', 'B', 'C', 'D', 'E']
['H', 'G', 'F', 'E', 'D', 'C', 'B', 'A']
```

```
sliced = list[1:5:2]
print(sliced)
```

```
['B', 'D']
```

List Methods

Function	Description
Append()	Add an element to the end of the list
Extend()	Add all elements of a list to the another list
Insert()	Insert an item at the defined index
Remove()	Removes an item from the list
Pop()	Removes and returns an element at the given index

2nd Topic

Tuple

Tuple

Tuple is similar to a list. It can store any data type but it can't be mutated - the elements inside can't be changed,

```
Tuple = (0, 1, 2) Tuple = (0, 1, 2)   Tuple = (0, 1, 2)
Tuple[0] = 5 Tuple = (0, 1, 2)   Tuple = (0, 1, 2)
```

Exception has occurred: TypeError ×

```
'tuple' object does not support item assignment
```

```
Tuple = (0, 1, 2)
Tuple = (5, 1, 2)
```

Creating a Tuple

```
# Tuple
Tuple = ()
Tuple = (1, 2, 3)
Tuple = ("Tuple", 123)
list = [1, 2, 3, 4, 5]
print(tuple(list))
```

```
(1, 2, 3, 4, 5)
```

Accessing Tuple

```
Tuple = ("A", "B", "C")
print(Tuple[1])
```

```
B
```

```
a, b, c = Tuple
print(a)
print(b)
print(c)
```

```
A
B
C
```

Concatenation of Tuples

```
Tuple1 = (0, 1, 2)
Tuple2 = Tuple1 + ("A", "B", "C")
print(Tuple2)
```

```
(0, 1, 2, 'A', 'B', 'C')
```

3rd Topic

Sets

Set

Sets are an unordered and unindexed collection of data type that are iterable, mutable and have no duplicate elements.

No duplicate (If Integers will order in ascending)

```
Set = {5, 3, 2, 3, 2, 1}
print(Set)
```

```
{1, 2, 3, 5}
```

Unordered

```
Set = {"A", "B", "C"}
print(Set) # This will print randomly
```

```
{'B', 'C', 'A'}
{'A', 'C', 'B'}
```

Unindexed

```
Set[0] Set = {1, 2, 3, 5}   Set = {1, 2, 3, 5}
```

Exception has occurred: TypeError ×

```
'set' object is not subscriptable
```

Iterable

```
Set = {5, 3, 2, 3, 2, 1} # {1, 2, 3, 5}
for val in Set:
    print(val)
```

```
1
2
3
5
```

Creating a set

```
Set = {3, 3, 2, 1}
Set = set("SetsAreConfusing")
print(Set)
Set = set([5, 5, 4, 3, 7])
print(Set)
```

```
{'s', 'u', 'A', 'e', 'f', 'C', 'r', 'n', 'o', 't', 'g', 'i', 'S'}
{3, 4, 5, 7}
```

Adding element to a set

```
Set = set()
Set.add(3) # Only used on hashable / immutable objects
Set.add((2, 6))
print(Set)
```

```
{3, (2, 6)}
```

```
Set = {1,99,22}
Set.update([10, 22])
print(Set)
```

```
{1, 10, 99, 22}
```

Accessing a set

```
Set = {"List", "Tuple", "Set"}
print(Set)

for i in Set:
    print(i)

print("List" in Set)
```

```
{'Tuple', 'List', 'Set'}
Tuple
List
Set
```

```
True
```

Removing elements from a set

```
{'A', 'C', 'D'}
```

```
Set = {"A", "B", "C", "D"}
Set.discard("B")
print(Set)
```

```
removed = Set.pop() # Returns and removes a random element
print(removed)
print(Set)
```

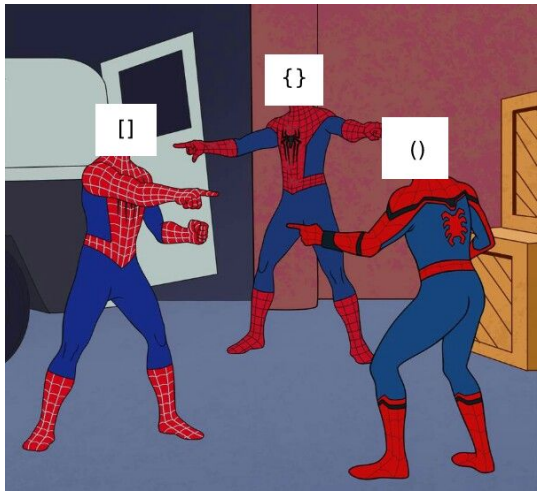
```
A
{'C', 'D'}
```

```
Set.clear()
print(Set)
```

```
set()
```

Differences between lists, tuples and sets

List	Set	Tuple
Lists is Mutable	Set is Mutable	Tuple is Immutable
It is Ordered collection of items	It is Unordered collection of items	It is Ordered collection of items
Items in list can be replaced or changed	Items in set cannot be changed or replaced	Items in tuple cannot be changed or replaced



4th Topic

Dictionaries

Dictionary

Dictionary is an ordered collection of data values, used to store data values like a map. Unlike other Data Types that hold only a single value as an element, Dictionary holds **key:value** pair.

```
myDict = {
    #Key         #Value
    "Subject": "Python",
    "Level": "Intermediate",
    "ID": 123456,
    555: "UwU"
}
```

Creating a Dictionary

```
# Create a dictionary
# Dictoinary with string keys
Dict = {"key":"value",
        "key1": "value1",
        "key2": "value2"}
print(Dict)
```

```
{'key': 'value', 'key1': 'value1', 'key2': 'value2'}
```

```
# Dictionary with mixed keys
Dict = {0:"value",
        1:"value1",
        "key2": "value2"}
print(Dict)
```

```
{0: 'value', 1: 'value1', 'key2': 'value2'}
```

```
# Nested dictionary
Dict = {"key": "value",
        "key1": "value1",
        "key2": {"nested_key": "nested_value",
                 "nested_key1": "nested_value1"}}
print(Dict)
```

```
{'key': 'value', 'key1': 'value1', 'key2': {'nested_key': 'nested_value', 'nested_key1': 'nested_value1'}}
```

Accessing a Dictionary

```
# Accessing elements from a Dictionary
Dict = {0: "value",
        1: "value1",
        "key2": "value2"}
print(Dict[0])
print(Dict["key2"])
```

```
value
value2
```

```
# Accessing elements from a Nested-Dict
Dict = {"key": "value",
        "key1": "value1",
        "key2": {"nested_key": "nested_value",
                 "nested_key1": "nested_value1"}}
print(Dict["key2"]["nested_key"])
```

```
nested_value
```

Adding element to a Dictionary

```
# Adding an element to a Dictionary
Dict = {}
Dict[0] = "value"
Dict[1] = "value1"
Dict[2] = "value2"
print(Dict)
```

```
{0: 'value', 1: 'value1', 2: 'value2'}
```

```
# Update existing ket's value
Dict[2] = "updated"
print(Dict)
```

```
{0: 'value', 1: 'value1', 2: 'updated'}
```

Removing elements from a Dictionary

```
# Removing elements from Dictionary
# del
Dict = {"key":"value",
        "key1": "value1",
        "key2": "value2"}
print(Dict)
del Dict["key"]
print(Dict)
```

```
{'key': 'value', 'key1': 'value1', 'key2':
{'key1': 'value1', 'key2': 'value2'}
```

```
# Pop
Dict = {"key":"value",
        "key1": "value1",
        "key2": "value2"}
print(Dict)
pop_ele = Dict.pop("key")
print(Dict)
print(pop_ele)
```

```
{'key': 'value', 'key1': 'value1', 'key2': '
{'key1': 'value1', 'key2': 'value2'}
value
```

5th Topic

Exception Handling

The difference between Syntax Error and Exceptions

```
#syntax error
num = 100
if(num > 10)
    print("Correct!")
```

```
    if(num > 10)
                ^
SyntaxError: invalid syntax
```

```
#exceptions
num = 100
a = num/0
print(a)
```

```
250  #exceptions
251  num = 100 num = 100   num = 100
252  a = num/0 num = 100   num = 100
```

Exception has occurred: ZeroDivisionError ×
division by zero

Common Exceptions

`IndexError`	Raised when the index of a sequence is out of range.
`KeyError`	Raised when a key is not found in a dictionary.
`NameError`	Raised when a variable is not found in local or global scope.
`TypeError`	Raised when a function or operation is applied to an object of incorrect type.
`ValueError`	Raised when a function gets an argument of correct type but improper value.
`ZeroDivisionError`	Raised when the second operand of division or modulo operation is zero.

Common Exceptions

`IndexError` — Raised when the index of a sequence is out of range.

```
1
2   list = [0,1,2,3] list = [0, 1, 2, 3]   list = [0, 1, 2, 3]
3   list[4] list = [0, 1, 2, 3]   list = [0, 1, 2, 3]
```

Exception has occurred: IndexError ×
`list index out of range`

`KeyError` — Raised when a key is not found in a dictionary.

```
1
2   Dict = {"key1": "value1"} Dict = {'key1': 'value1'}   Dict = {'key1': 'value1'}
3   Dict["key2"] Dict = {'key1': 'value1'}   Dict = {'key1': 'value1'}
```

Exception has occurred: KeyError ×
`'key2'`

Common Exceptions

`NameError`	Raised when a variable is not found in local or global scope.

```
1
2   a = 5 a = 5   a = 5
3   b = 7 b = 7   b = 7
4   print(c)
```

Exception has occurred: NameError ×

```
name 'c' is not defined
```

Common Exceptions

`TypeError`	Raised when a function or operation is applied to an object of incorrect type.

```
1
2  print("String" + 5)
```

Exception has occurred: TypeError ×
`can only concatenate str (not "int") to str`

`ValueError`	Raised when a function gets an argument of correct type but improper value.

```
1
2  userInput = int(input("Enter Int: "))
```

Exception has occurred: ValueError ×
`invalid literal for int() with base 10: 'No'`

```
try:
        # Some Code....

except:
        # optional block
        # Handling of exception (if required)

else:
        # execute if no exception

finally:
      # Some code .....(always executed)
```

Except Everything

```
def divide(a, b):
    try:
        result = a / b
        print("Result: ", result)
    except:
        print("Error")

divide(3, 2)
divide(3, 0)
```

```
Result:  1.5
Error
```


```
Result:  1.5
Error
```

Except Specific Error

```
def divide(a, b):
    try:
        result = a / b # Exception occurs here so it skips the next line
        print("Result: ", result)
    except ZeroDivisionError:
        print("Cannot divide by zero")
```

```
divide(3, 2)
divide(3, 0)
```

```
Result:  1.5
Cannot divide by Zero
```

Print Exception

```
def divide(a, b):
    try:
        result = a / b
        print("Result: ", result)
    except Exception as e:
        print("Error: ", e)
```

```
divide(3, 2)
divide(3, 0)
```

```
Result:  1.5
Error:  division by zero
```

Else

```
def divide(a, b):
    try:
        result = a / b
    except ZeroDivisionError:
        print("Cannot divide by zero")
    else:
        print("Result: ", result)
```

```
divide(3, 2)
divide(3, 0)
```

```
Result:  1.5
Cannot divide by zero
```

6th Topic

Recursion

```
def factorial(n):
        if (n == 1):
                return 1
        else:
                return n * factorial(n - 1)
```

```
def fact(n):
        total = 1
        for i in range (1, n + 1):
                total *= i
        return total
```

function(n)	Operation
factorial(5)	5 * factorial(4)
factorial(4)	4 * factorial(3)
factorial(3)	3 * factorial(2)
factorial(2)	2 * factorial(1)
factorial(1)	1 * factorial(0)
factorial(0)	1

```
def factorial(n):
        if (n == 1):
                return 1
        else:
                return n * factorial(n - 1)
```

```
def fact(n):
        total = 1
        for i in range (1, n + 1):
                total *= i
        return total
```

Function	Operation	Return value
factorial(5)	5 * factorial(4)	5 * 24 = 120
factorial(4)	4 * factorial(3)	4 * 6 = 24
factorial(3)	3 * factorial(2)	3 * 2 = 6
factorial(2)	2 * factorial(1)	2 * 1 = 2
factorial(1)	1	1

```
# Fibonacci Sequence
# 1 1 2 3 5 8 13 21 34 ...

def fib(n):
        if (n <= 2):
                return 1
        else:
                return fib(n-1) + fib(n-2)
```

Function	Operation	Return
fib(6)	fib(5) + fib(4)	5 + 3 = 8
fib(5)	fib(4) + fib(3)	3 + 2 = 5
fib(4)	fib(3) + fib(2)	2 + 1 = 3
fib(3)	fib(2) + fib(1)	1 + 1 = 2
fib(2)	1	1
fib(1)	1	1