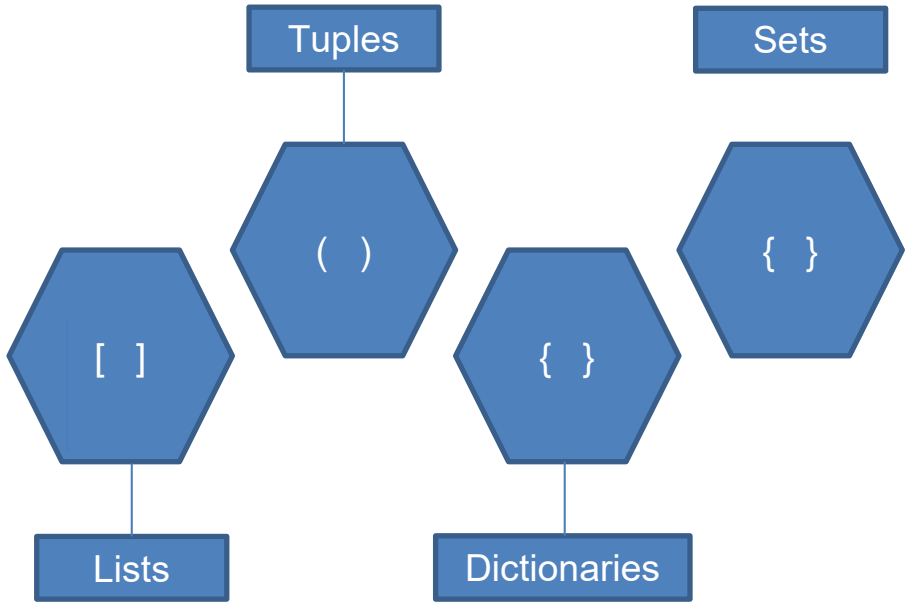
PYTHON

Data Structures and Functions

TABLE OF CONTENT

PYTHON DATA STRUCTURES

- Used to store and organize the data



LISTS

- Most widely used data structures
- Mutable
- Used to store an ordered collection of items


```
# List declaration - 1
List_1  = [1, "a#" , "Apple" , 2-2] # syntax
print("List 1:",List_1)

# List declaration - 2
a = "Variable"

string1 = "This is a string"

operation = int(18-12/3)    # try type casting to < int, float, str >

List_2  = [1, a , string1 , int(operation/4)]   # try type casting to < int, float, str >

print("\nList 2:", List_2)
```

```
List 1: [1, 'a#', 'Apple', 0]

List 2: [1, 'Variable', 'This is a string', 3]
```

LIST METHODS

- append()
- sort()
- extend()
- pop()
- insert()
- clear()
- index()
- remove()
- count()
- reverse()
- copy()

```
# Declaring a list
List  = [1, "a#" , "python" , 2-2]
print(List)

# add 6 to the above list
List.append(6)
print(List)

List.extend([2,"Python 3.8 current version", 4])
print(List)

# pop deletes the last element of the list
List.pop()   # put a inyeger followed by "-" to pop a value
print(List)

# complicated way to print a list
for i in range(len(List)):
    print(List[i])
```

```
[1, 'a#', 'python', 0]
[1, 'a#', 'python', 0, 6]
[1, 'a#', 'python', 0, 6, 2, 'Python 3.8 current version', 4]
[1, 'a#', 'python', 0, 6, 2, 'Python 3.8 current version']
1
a#
python
0
6
2
Python 3.8 current version
```

TUPLES

- Sequence data type
- Immutable
- Tuple unpacking is also possible

```
# Create a new tuple
new_tuple = (1, "#$a", "Python", 1+2)
print(new_tuple)
print("\n",new_tuple[1])
```

```
(1, '#$a', 'Python', 3)

 #$a
```

```
# Tuple cannot be changes. The below code will give error, soplease don't be alarmed :)
new_tuple[1] = 0
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\Users\GUZAYY~1\AppData\Local\Temp/ipykernel_55568/1852451153.py in <module>
      1 # Tuple cannot be changes. The below code will give error, soplease don't be alarmed :)
----> 2 new_tuple[1] = 0

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

TUPLE METHODS

- index()
- count()

```
In [11]: new_tuple = (1, "#$a", "Python")
         print(new_tuple.index("Python"))
```

```
2
```

```
In [14]: new_tuple = (1, "#$a", "Python", 1 , 1, 3.5, 1)
         print(new_tuple.count(1))
```

```
4
```

```
In [15]: #tuple unpacking is also possible
         new_tuple = (1, "#$a", "Python")
         a, b, c = new_tuple
         print(a)
         print(b)
         print(c)
```

```
1
#$a
Python
```

DICTIONARY

- Key value pairs
- Dictionaries are indexed by keys
- Keys are unique (within one dictionary)
- Main operation on a dictionary is storing a value with some key and extracting the value given the key

```
# Create a new dictionary
dictionary = dict() # or d = {}

# Add a key - value pairs to dictionary
dictionary['One'] = 1
dictionary['2'] = 2
dictionary['B']=3.5

# print the whole dictionary
print(dictionary)

# print only the keys
print("\n",dictionary.keys())

# print only values
print("\n",dictionary.values())
```

```
{'One': 1, '2': 2, 'B': 3.5}

 dict_keys(['One', '2', 'B'])

 dict_values([1, 2, 3.5])
```

DICTIONARY METHODS

- clear()
- copy()
- items()
- fromkeys()
- get()
- keys()
- pop()
- popitem()
- update()
- values()
- setdefault()

```
my_dict = {'name': 'Jack', 'age': 26}
print(my_dict['name'])
print(my_dict.get('age'))
```

```
Jack
26
```

```
# Trying to access keys which doesn't exist throws error
print(my_dict.get('address'))
print(my_dict['address'])
```

```
None

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
C:\Users\GUZAYY~1\AppData\Local\Temp/ipykernel_55568/2116744555.py in <module>
      1 # Trying to access keys which doesn't exist throws error
      2 print(my_dict.get('address'))
----> 3 print(my_dict['address'])

KeyError: 'address'
```

SETS

- An unordered collection data type
- Iterable, mutable
- Has no duplicate elements
- Cannot access items using indexes
- **Frozen sets** in Python are immutable objects
- Frozen set only support methods and operators that produce a result without affecting the frozen set

Normal vs Frozen Set

```
# Same as {"a", "b","c"}
normal_set = set(["a", "b","c"])

print("Normal Set")
print(normal_set)

# A frozen set
frozen_set = frozenset(["e", "f", "g"])

print("\nFrozen Set")
print(frozen_set)
```

```
Normal Set
{'a', 'c', 'b'}

Frozen Set
frozenset({'g', 'e', 'f'})
```

SET METHODS

- add()
- union()
- intersection()
- difference()
- clear()

Set methods

```
# Union of sets

ml = {"Regression", "Classifier"}
dl = {"Computer vision", "NLP"}
language = {"Python", "R"}

# Union using union() function
data_science = new_set.union(dl)

print("Union using union() function")
print(data_science)
```

```
Union using union() function
{'NLP', 'Computer vision'}
```

COMPREHENSIONS (WITH LIST, DICTIONARY, SET)

- A short and concise way to construct new sequences (such as lists, set, dictionary etc.) using sequences which have been already defined
- List comprehension syntax:
  output_list =
  [output_exp for var in input_list if (var satisfies this condition)]
- Dictionary comprehension syntax:
  output_dict = { key: value for (key, value) in Iterable if (key, value satisfy this condition)}

```
# Using List comprehensions
list_1 = [1, 2, 3, 4, 4, 5, 6, 7, 7]
output_list = []

# Using loop for constructing output list
for value in list_1:
    if value % 2 == 0:
        output_list.append(value)

print("Output List using for loop is :", output_list)
```

```
Output List using for loop is : [2, 4, 4, 6]
```

CONDITIONAL STATEMENTS

- Decision making statements
- Used to control program flow
- At runtime, used for decision making operations
- Intended block is executed only “if” condition is True
- Otherwise optional “else” block is executed

if, elif, else statements

```
name = 'Joe'
if name == 'Fred':
    print('Hello Fred')
elif name == 'Xander':
    print('Hello Xander')
elif name == 'Joe':
    print('Hello Joe')
elif name == 'Arnold':
    print('Hello Arnold')
else:
    print("I don't know who you are!")
```

```
Hello Joe
```

One liner if, elif, else statements

```
num = 2
if num == 1: print('my'); print('python'); print('version')
elif num == 2: print('python-version'); print('3.8')
else: print('version'); print('mismatch')
```

```
python-version
3.8
```

LOOPING STATEMENTS

- Used to program repetitive tasks
- While:- keep repeating the task while it is not done (condition)
- For:- for this condition keep repeating the task until it is done

While loop

```
# Take input from user
input_num = int(input("Enter n: "))
sum = 0
num = 1

while num <= input_num:
    sum = sum + num
    num+=1    # update counter

# print the sum
print("The sum is", sum)
```

```
Enter n: 10
The sum is 55
```

For loop

```
languages = ['R', 'Python',  'Scala', 'Java', 'C++']

for index in range(len(languages)):
   print('Current language:', languages[index])
```

```
Current language: R
Current language: Python
Current language: Scala
Current language: Java
Current language: C++
```

PYTHON FUNCTIONS

- Functions in real life:- running, cooking
- Function in programming:- function to compute area of a circle, function to deposit money, etc.
- Functions break program into smaller and modular chunks
- Makes larger programs more organized manageable and reusable
- Optional documentation string (docstring) to describe what the function does

PYTHON FUNCTIONS: IN-BUILT FUNCTIONS

- There are many inbuilt functions in python

Function	Description
abs()	Returns the absolute value of a number
all()	Returns True if all items in an iterable object are true
any()	Returns True if any item in an iterable object is true
ascii()	Returns a readable version of an object. Replaces none-ascii characters with escape character
bin()	Returns the binary version of a number
bool()	Returns the boolean value of the specified object
bytearray()	Returns an array of bytes
bytes()	Returns a bytes object
callable()	Returns True if the specified object is callable, otherwise False
chr()	Returns a character from the specified Unicode code
classmethod()	Converts a method into a class method
compile()	Returns the specified source as an object, ready to be executed
complex()	Returns a complex number

PYTHON FUNCTIONS: USER DEFINED FUNCTIONS

- Define a function
- Specify the set of instructions to be performed
- Call the function whenever required

Function without arguments

```
def user_defined():
  print("Welcome to GreatLearning")

# Driver code to call a function
user_defined()
```

```
Welcome to GreatLearning
```

Function with arguments

```
def finding_even_number(num):
    if (num % 2 == 0):
        print("Given number is even")
    else:
        print("Given number is odd")


# Driver code to call the function
finding_even_number(2)
finding_even_number(3)
```

```
Given number is even
Given number is odd
```

SUMMARY

- Discussed data structures in Python
- Methods associated with data structures
- Comprehensions in Python
- Conditional statements and conditional expression (Iterary operator)
- Loops in Python
- Build-in and user-defined functions

Hands-On

THANK YOU

Happy Learning ☺