Day 6 Assignment: Amaan Shaikh

Python Day 1 Topics

Python Data Types

- **Numeric**: Integers (int), Floating-point numbers (float), Complex numbers (complex).
- **Sequence**: Strings (str), Lists (list), Tuples (tuple).
- **Boolean**: Represents True or False.
- **Set**: Unordered collection of unique elements (set).
- **Mapping**: Key-value pairs (dict).

Keywords, Identifiers, Operators

- **Keywords**: Reserved words in Python (e.g., if, else, for, while, def, class).
- **Identifiers**: Names used for variables, functions, classes, etc., following rules (e.g., my_var, calculateSum).

- **Operators**:
    - **Arithmetic**: +, -, *, /, //, %, **
    - **Comparison**: ==, !=, <, >, <=, >=
    - **Logical**: and, or, not
    - **Assignment**: =, +=, -=, *=, /=
    - **Bitwise**: &, |, ^, ~, <<, >>

Feature	**List**	**Tuple**	**Dictionary**	**Set**
Can Store Different Types	Yes	Yes	Yes (keys and values)	Yes
Mutable	Yes	No	Yes (values), keys are immutable	Yes
Access Type	Index-based	Index-based	Key-based	No direct indexing
Allows Duplicates	Yes	Yes	Keys: No, Values: Yes	No
Ordered	Yes	Yes	Yes	No
Syntax	[1, 2, 3]	(1, 2, 3)	{'key1': 'value1', 'key2': 'value2'}	{1, 2, 3}

Imp Functions:

Data Type	**Function/Method**	**Description**	**Example Usage**
List	append()	Adds an element to the end of the list.	my_list.append(4)
	extend()	Extends the list by appending elements from an iterable.	my_list.extend([5, 6])
	insert()	Inserts an element at a specific position.	my_list.insert(1, 'apple')
	remove()	Removes the first occurrence of an element.	my_list.remove(2)
	pop()	Removes and returns the element at the specified position (default is the last).	my_list.pop(2)

	clear()	Removes all elements from the list.	my_list.clear()
	index()	Returns the index of the first occurrence of an element.	my_list.index('apple')

Data Structure	**Function/Method**	**Description**	**Example Usage**
Tuple	count(x)	Returns the number of times x appears in the tuple.	t = (1, 2, 2, 3); t.count(2)
	index(x)	Returns the index of the first occurrence of x.	t = (1, 2, 3); t.index(3)
	len()	Returns the length of the tuple.	t = (1, 2, 3); len(t)
	min()	Returns the smallest item in the tuple.	t = (1, 2, 3); min(t)
	max()	Returns the largest item in the tuple.	t = (1, 2, 3); max(t)
	sum()	Returns the sum of all items in the tuple.	t = (1, 2, 3); sum(t)
	sorted()	Returns a sorted list of the tuple's items.	t = (3, 1, 2); sorted(t)

Dictionaries:

Function/Method	**Description**	**Example Usage**	**Output**
dict()	Creates a new dictionary.	d = dict(a=1, b=2)	{'a': 1, 'b': 2}
d.keys()	Returns a view of the dictionary's keys.	d.keys()	dict_keys(['a', 'b'])
d.values()	Returns a view of the dictionary's values.	d.values()	dict_values([1, 2])
d.items()	Returns a view of the dictionary's key-value pairs.	d.items()	dict_items([('a', 1), ('b', 2)])
d.get(key, default)	Returns the value for the specified key, or a default value if the key is not found.	d.get('a', 0)	1
d.update(other)	Updates the dictionary with key-value pairs from another dictionary or iterable.	d.update({'c': 3})	{'a': 1, 'b': 2, 'c': 3}
d.pop(key, default)	Removes the specified key and returns its value. If the key is not found, it returns the default value.	d.pop('b', 'Not Found')	2
d.popitem()	Removes and returns the last inserted key-value pair as a tuple.	d.popitem()	('c', 3)
d.clear()	Removes all key-value pairs from the dictionary.	d.clear()	{}
d.copy()	Returns a shallow copy of the dictionary.	d.copy()	{'a': 1, 'c': 3}

Sets:

Function	**Description**	**Example Usage**	**Output**
add()	Adds an element to the set.	s.add(4)	s = {1, 2, 3, 4}
remove()	Removes a specified element from the set. Raises KeyError if the element is not found.	s.remove(2)	s = {1, 3}
clear()	Removes all elements from the set.	s.clear()	s = set()
copy()	Returns a shallow copy of the set.	s_copy = s.copy()	s_copy = {1, 2, 3}
union()	Returns a new set containing all unique elements from the sets.	s1.union(s2)	s1 = {1, 2}, s2 = {2, 3}, s1.union(s2) = {1, 2, 3}
intersection()	Returns a new set with elements common to all sets.	s1.intersection(s2)	s1 = {1, 2, 3}, s2 = {2, 3, 4}, s1.intersection(s2) = {2, 3}

difference()	Returns a new set with elements in the first set that are not in the other sets.	s1.difference(s2)	s1 = {1, 2, 3}, s2 = {2, 3, 4}, s1.difference(s2) = {1}
symmetric_difference()	Returns a new set with elements in either set but not in both.	s1.symmetric_difference(s2)	s1 = {1, 2}, s2 = {2, 3}, s1.symmetric_difference(s2) = {1, 3}
issubset()	Checks if all elements of the set are in another set.	s1.issubset(s2)	s1 = {1}, s2 = {1, 2}, s1.issubset(s2) = True
issuperset()	Checks if all elements of another set are in the set.	s1.issuperset(s2)	s1 = {1, 2}, s2 = {1}, s1.issuperset(s2) = True

Math and String Functions

- **Math Functions**:
  - math.sqrt(x): Square root of x.
  - math.pow(x, y): x raised to the power of y.
  - math.ceil(x): Smallest integer greater than or equal to x.
  - math.floor(x): Largest integer less than or equal to x.

    - math.factorial(x): Factorial of x.
- **String Functions**:
    - str.lower(): Converts to lowercase.
    - str.upper(): Converts to uppercase.
    - str.strip(): Removes whitespace from both ends.
    - str.split(): Splits string into a list.
    - str.replace(old, new): Replaces occurrences of old with new.

Conditional Statements

- **if**: Evaluates a condition; executes block if condition is True.

```
if condition:
   # code
```

- **if-else**: Executes one block if condition is True, another if False.

```
if condition:
   # code
else:
   # code
```

- **if-elif-else**: Multiple conditions checked sequentially.

```
if condition1:
   # code
elif condition2:
   # code
else:
    # code
```

Loops:

- **for Loop**: Iterates over a sequence (list, tuple, dictionary, set, string).

```
for item in iterable:
    # code
```

- **while Loop**: Repeats as long as a condition is True.

```
while condition:
    # code
```

Control Statements:

- **break**: Exits the nearest enclosing loop.

```
for item in iterable:
    if condition:
        break
```

- **continue**: Skips the rest of the code inside the loop for the current iteration and proceeds to the next iteration.

```
for item in iterable:
    if condition:
        continue
    # code
```

- **pass**: A null statement used as a placeholder for future code.

```
if condition:
    pass
```