RAHUL
Date
Page 01

2300311016110 51

ASSIGNMENT - 2

Ques. Define what a variable is in Python and the rules for naming them and give examples of numeric data types in Python: int, float, and complex.

Soln: → A variable acts as a named storage location that holds data which can be manipulated during program execution. Unlike statically-typed languages, variables don't require explicit type declaration. The datatype is dynamically inferred from the assigned value.

The naming conventions enfore specific rules:

(i) It must start with a letter (a to z, or A to Z), or the underscore character.

(ii) A variable name cannot start with a number.

(iii) A variable can only contain alpha-numberic characters and underscore (A to Z, a to z, 0 to 9, and _).

(iv) variable names are case-sensitive (age, Age, and AGE are three different varidbles).

(v) A variable name cannot be any of the Python keywords.

Examples of three primary numeric data types supported by Python:-

(i) Integers (int): whole numbers without decimal

2300311016110 51

points, including negative values.
Ex: 'temperature = -5'

(ii) Floating-point (float): Real numbers with decimal precision.
Ex: 'pi = 3.14159'

(iii) Complex Numbers: It is expressed as 'a+bj', where 'a' is the real part, and 'b' is the imaginary part.
Ex: 'circuit-imp = 3+4j'

Ques. Explain the role of indentation in Python code.

Soln: Python employs mandatory indentation as structural syntax rather than just code formatting. Unlike C-style languages using braces {}, Python uses consistent whitespaces (typically 4 spaces, or one tab-space, or earlier 2 spaces were also used) to delimit code blocks. Whatever spacing you use, it must be consistent throughout the code. Improper indentation raises 'IndentationError', preventing execution.

Example of using indentation in Python:-

```
if x>10:
    print()
    if x<20:
        print()
else:
    print()
```

2300311016110 51

Ques. Explain string operations in python and how to manipulate strings with python script.

Soln: → Python strings are immutable sequences of Unicode characters supporting extensive manipulation through built-in methods and operators. Common operations include:-

i) concatenation:

```
'full-name = first + " " + last'
```

ii) Repetition:

```
'seperator = "-" * 30'
```

iii) Slicing:

```
domain = email[email.index('@')+1:]
```

iv) upper() : converts all characters in string to uppercase.

v) lower(): converts all characters in string to lowercase.

vi) find(substring): returns the lowest index in the string where the substring is found.

vii) strip(): removes any leading and trailing characters (space is default).

2300311016 11051

viii) replace (old, new) : replaces occurences of a substring within the string.

ix) split (delimeter) : splits the string at the specified delimeter and returns a list of substrings.

x) join (iterable) : concatenates elements of an iterable with a specified seperator.

xi) startswith (prefix) : checks if the string starts with a specified prefix.

xii) endswith (prefix) : checks if the string ends with the specified suffix.

∴ Example to use these in a python script :-

```
name = "Sarthak Sanay"
name.upper()
name.lower()
name.find("ar")          → 1
name.strip("S")
name.replace("a", "k")
name.split(" ")
name.startswith("S")     → True
name.endswith("na")      → False
```

we can also manipulate strings using the formatting strings options, given through f-strings.

2300311016051

∴ For example;

```
price = 49.9978
print(f"Total: ${price:.2f}")      # 49.99
```

Ques. Discuss the differences between lists and tuples in python with their use cases, and explain the different operations performed with python script.

Soln:→

	List	Tuple
i)	Lists are mutable.	Tuples are immutable.
ii)	The implication of iterations is time-consuming.	The implications of iterations is comparitively faster.
iii)	The list is better for performing operations, such as insertion and deletion.	A tuple data type is appropriate for accessing the elements.
iv)	Lists consume more memory.	Tuples consumes less memory as compared to the list.
	Lists have several built-in methods.	Tuples does not have any built-in methods.
	unexpected changes and errors are more to occur.	Because tuples don't change they are far less error-prone.

2300311016111051

(i) Use cases for lists :-

The lists are mutable, therefore they are typically used when the collection of items is expected to change over time.

∴ For example :-

```
tasks = ["buy chocolates", "study"]
tasks.append("go to gym")
tasks.remove("study")

stack = []
stack.append("First action")
stack.append("second action")
stack.pop()      # Remove ~~or~~ second action

mixed_data = [1, "apple", 3.14, ["nested", "list"]]
```

(ii) Use cases for Tuples :-

Since, tuples are immutable, they are better suited for scenarios where you need to ensure that the data should not change after creation. Tuples are often used in situations where the integrity of the data must be preserved, or when performance optimization is important.

∴ For example :-

```
coordinates = (40.1728, -74.0060)  #Immutable,
                                    fixed pair of
                                    coordinates
```

2300311016l1051

```
location_data = {}                    # Tuple as key
location_data[(40.1728, -74.0060)] = "New York"

point = (5,10)
x, y = point  #unpacking tuple into individual
                variables

data_points = [(1,2), (3,4), (5,6)]   # Tuples for
                                        fixed
                                        coordinates.
```

Ques. Explain the Dictionary data type in Python with its use cases, and explain the different operations performed with Python script.

Soln:→ In python, a dictionary is an unordered, mutable collection of key-value pairs. It allows you to store data in a way that associates a unique key with a value. This data structure is optimized for faster lookups.

Key characteristics of dictionaries:-

i) The order of items in a dictionary is unordered.
ii) Dictionaries are mutable. It can be modified.
iii) Each key in a dictionary is unique.
iv) Dictionary keys must be immutable types (like strings, numbers, or tuples). Lists or other dictionaries cannot be used as dictionary keys.

Examples of some use-cases of dictionaries are:-

i) To store key-value pairs for configuration settings.
ii) A dictionary can be used to count occurences of elements in a list (commonly, known as frequency count).
iii) Dictionaries are useful when you need to map one set of data to another. (e.g., mapping student names to their grades).
iv) To store an object's attributes as key-value pairs.
v) Storing and searching data by key, in situations where you need fast lookups, dictionaries are efficient.

∴ The operations you can perform on dictionaries are as follows:-

```
my_dict = {'name': 'sarthak', 'age': 19}
print(my_dict['name'])
print(my_dict.get('name'))
print(my_dict.get('address', 'Not available'))

print my_dict['location'] = 'New York'
my_dict['age'] = 26

del my_dict['age'].

age = my_dict.pop('age', 'key not found')

item = my_dict.popitem().
```

2300311016110 51

```
print('name' in my-dict)      # True
print('address' in my-dict)   # False

# Iterate over key, value and key-value pairs

for key in my-dict.keys():
    print(key)

for value in my-dict.values():
    print(value)

for key, value in my-dict.items():
    print(key, value)

# Merging dictionaries

dict1 = {'a':1 , 'b':2}
dict2 = {'c':3 , 'd':4}
dict1.update(dict2)          # method-1

merged-dict = dict1 | dict2   # method-2

# copying dictionaries

copy-dict = my-dict.copy()
```

Ques. Explain the set data type in Python with its use cases, and explain the different operations performed with python script.

2300311016110051

Soln:→ Set is an unordered collection of unique elements. sets are mutable, meaning you can add or remove items after their creation. They are often used for operations that require uniqueness, membership testing, and set mathematical operations (like unions, intersections, etc.)

Key characteristics of sets:-
i) The elements are unordered, and have no specific order.
ii) Sets cannot have ~~unique~~ duplicate elements. If you try to add a duplicate, it is ignored.
iii) You can add or remove items. It is mutable.
iv) Elements must be immutable (e.g. numbers, strings, tuples), but the set itself is mutable.

Syntax to create a set:-
```
my_set = {1, 2, 3}
empty_set = set()
```

Use cases of sets:-
(i) To eliminate duplicates from a collection, you can convert a list to a set.
(ii) sets allow fast membership testing to check if an item exists.
(iii) Supports set operations like union, intersection and difference.

Set operations are as follows:-

2300311016111051

```
my_set = {1, 2, 3}
my_set.add(4)
my_set.remove(2)
my_set.discard(3)
popped_item = my_set.pop()
my_set.clear()

set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1 | set2)      # Union
print(set1 & set2)      # Intersection
print(set1 - set2)      # Difference
print(set1 ^ set2)      # Symmetric Difference

set1 = {1, 2}
set2 = {1, 2, 3}
print(set1.issubset(set2))    # True
print(set2.issuperset(set1))  # True

set1 = {1, 2}
set2 = {3, 4}
print(set1.isdisjoint(set2))  # True

new_set = set1.copy()

print(len(set1))      # 2

print(2 in set1)      # True
print(3 in set1)      # False
```

2300311016110 51

Quer. List and explain the different types of operators in Python with script (arithmetic, relational, logical, etc.)

Soln:→ 1) Arithmetic operators :- (+, -, /, *, %, //, **)

Arithmetic operators (+, -, /, *, %) are used to perform basic mathematical operations like addition, multiplication, subtraction, division, etc. To get integer result, use floor-division (//).

```
a = 15
b = 4
print(a+b)      # 19      | Addition
print(a-b)      # 11      | Subtraction
print(a*b)      # 60      | multiplication
print(a/b)      # 3.75    | Division
print(a//b)     # 3       | Floor Division
print(a%b)      # 3       | modulus
print(a**b)     # 50625   | Exponentiation.
```

2) Relational Operators :- (>, <, >=, <=, ==, !=)

To compare values, we use relational operators. It either returns 'True' or 'False' according to the condition.

```
a = 13
b = 33
print(a>b)      | False
print(a<b)      | True
print(a==b)     | False
```

2300311016110 51

print(a != b)	True
print(a >= b)	False
print(a <= b)	True

3) Logical operators:- (and, or, not)
To perform logical operations, in Python, we ~~per~~ use Logical AND, Logical OR, Logical NOT. It is used to combine conditional statements.

a = True
b = False

print(a and b)	False
print(a or b)	True
print(not a)	False

4) Bitwise Operators:- (NOT, Shift, AND, XOR, OR)
These operators act on bits and perform bit-by-bit operations.

a = 10
b = 4

print(a & b)	0	Logical AND
print(a | b)	14	Logical OR
print(~a)	-11	Logical NOT
print(a ^ b)	14	Logical XOR
print(a >> 2)	2	Right shift
print(a << 2)	40	Left shift

2300311016111051

5) Assignment operators :- (=, +=, -=, *=, /=, <<=)
These operators are used to assign values to the variables. This operator is used to assign the value of the right side of the expression to the left side of the operand section.

a = 10	
b = a	10
b += a	20
b -= a	10
b *= a	100
b <<= a	102400

6) Identity operators :- (is, is not)
In python, 'is' and 'is not' are the identity operators, both are used to check if two values are located on the same part of the memory. Two variables that are equal do not imply that they are identical.

is : True if the operands are identical.
is not : True if the operands are not identical.

```
a = 10
b = 20
c = a

print(a is not b)   #True
print(a is c)       #True
```

2300311016110 51

7) membership operators in python :- (in, not in)
In python, 'in' and 'not in' are the membership operators that are used to test whether a value or variable is in a sequence.

in : True if value is found in the sequence.
not in : True if value is not found in the sequen

```
x = 24
list = [10, 20, 30, 40, 50]
if (x not in list):
    print("x is NOT in the list.")
else:
    print ("x is given in the list.")
```

8) conditional/Ternary operators :-
Ternary operators are also known as conditional expressions are operators that evaluate something based on condition being true or false.
It simply allows testing a condition in a single line replacing the multi-line if-else making the code compact.

Syntax: [on_true] if [expression] else [on_false]

Examples:-

```
a, b = 10, 20
min = a if a<b else b
print(min)
```

10

2300311016111051

Ques. Describe operator precedence and how it affects expressions in Python.

Soln: → In python, operators have different levels of precedence, which determine the order in which they are evaluated. when multiple operators are present in an expression, the ones with higher precedence are evaluated first. In the case of operators with the same precedence, their associativity comes into play, determining the order of evaluation.

Precedence	Operators	Description	Associativity
1	()	Paranthesis	L to R
2	x[index], x[index:index]	subscription, slicing	L to R
3	await x	await exp.	N/a
4	**	Exponentiation	R to L
5	+x, -x, ~x	+ve, -ve, bitwise NOT	R to L
6	*, @, /, //, %	arithmetic	L to R
7	+, -	add & subtract	L to R
8	<< , >>	shifts	L to R

2300311016111051

Precedence	Operators	Description	Associativity
9.	&	Bitwise AND	L TO R
10.	^	Bitwise XOR	L to R
11.	|	Bitwise OR	L to R
12.	in, not in, is, ~~not in~~ is not, <, <=, >, >=, !=, ==	comparisions, membership tests, identity tests	L to R
13.	not x	Boolean NOT	R to L
14.	and	Boolean AND	L to R
15.	or	Boolean OR	L to R
16.	if-else	conditional expression	R to L
17.	lambda	Lambda expression	N/A
18.	:=	Assignment expression (walrus operator)	R to L

2300311016110 51

Operator precedence determines the order in which operators are evaluated. It's important because, without understanding it, the result of complex expressions might not be what you expect. when evaluating an expression with multiple operators, the python interpreter follows a specific order. operators with higher precedence are evaluated before operators with lower precedence.

For example:-

```
calculation = 5 + 3 * 2 ** 2 / 4
# Evaluates as 5 + (3 * (2 ** 2)) / 4  → 5 + (3 * 4) / 4
#                                      → 5 + (12 / 4)
#                                      → 5 + 3
#                                      → 8
```