TYPE-A:DATA HANDLING:CH-7

1) What are data types in Python? How are they important?

sol:

Data types in Python tell the type of data a variable can store, such as numbers, text, or true/false values.

They are important because they help Python understand how to store and process data correctly. Data types make programs more efficient and reduce errors.

2) How many integer types are supported by Python? Name them.

sol:

Python supports only one integer type.

It is:

int

3) How are these numbers different from one another (with respect to Python)? 33, 33.0, 33j, 33 + j

sol:

Number	Type in Python	Meaning
33	int	An integer number
33.0	float	A decimal (floating point) number
33j	complex	An imaginary number with real part 0
33 + j	complex	A complex number with real part 33 and imaginary part 1

4) The complex numbers have two parts : real and imaginary. In which data type are real and imaginary parts represented ?

sol:

In Python, the real and imaginary parts of a complex number are represented using the float data type.

5) How many string types does Python support? How are they different from one another?

sol:

String Type	Representation	Difference
Single-quoted String	'Hello'	Written using single quotes
Double-quoted String	"Hello"	Written using double quotes
Triple-quoted String	'''Hello''' or """Hello"""	Used for multi-line strings

6) What will following code print?

```
str1 = '''Hell
o'''
str2 = '''Hell\
o'''
print(len(str1) > len(str2))
```

sol:
True

7) What are Immutable and Mutable types in Python? List immutable and mutable types of Python.

sol:

Type	Meaning	Examples
Immutable	Values cannot be changed in place	int, float, complex, str, tuple
Mutable	Values can be changed in place	list, dict, set

8) What are three internal key-attributes of a value-variable in Python ? Explain with example.

sol:
In Python, every value stored in a variable has three internal key attributes:
1. Identity – It tells where the object is stored in memory.
2. Type – It tells what kind of data the object is.
3. Value – It tells the actual data stored in the variable.

These three attributes can be checked using the functions id(), type(), and by printing the variable.

Example:

```
x = 10
print(id(x))    # Identity
print(type(x))  # Type
print(x)        # Value
```

9) Is it true that if two objects return True for is operator, they will also return True for == operator?

sol:
True

10) Are these values equal? Why/why not?

1. 20 and 20.0
2. 20 and int(20)
3. str(20) and str(20.0)
4. 'a' and "a"

sol:

Values	Are They Equal?	Reason
20 and 20.0	Yes	They have the same numeric value, so == gives True

20 and int(20)	Yes	int(20) is also 20
str(20) and str(20.0)	No	"20" and "20.0" are different strings
'a' and "a"	Yes	Both represent the same string

11) What is an atom in Python? What is an expression?

sol:
An atom in Python is the smallest unit of an expression. It represents a single value, name, or object.

An expression is a combination of atoms, variables, and operators that produces a result.

12) What is the difference between implicit type conversion and explicit type conversion?

sol:

Implicit Type Conversion	**Explicit Type Conversion**
It is done automatically by Python.	It is done manually by the programmer.
No function is needed.	Type conversion functions are used.
It reduces data loss.	It may cause data loss.
Example: int to float	Example: int(), float(), str()

13) Two objects (say a and b) when compared using == ,return True. But Python gives False when compared using is operator. Why? (i.e., a == b is True but why is a is b False?)

sol:
- The == operator checks whether two objects have the same value.
- The is operator checks whether two objects refer to the same memory location.
- So, even if a and b have the same value and a == b is True, a is b can be False because they are stored as different objects in memory.
- Therefore, == compares values, while is compares identity.

14) Given str1 = "Hello", what will be the values of:

(a) str1[0]
(b) str1[1]
(c) str1[-5]
(d) str1[-4]
(e) str1[5]

sol:
(a) str1[0] = H
(b) str1[1] = e
(c) str1[-5] = H
(d) str1[-4] = e
(e) str1[5] = Error (Index out of range)

15) If you give the following for str1 = "Hello", why does Python report error?
str1[2] = 'p'

sol:
Python reports an error for str1[2] = 'p' because strings in Python are immutable.
This means the characters of a string cannot be changed after the string is created.
So, assigning a new value to a string position is not allowed and causes an error.

16) What will the result given by the following?

(a) type (6 + 3)
(b) type (6 -3)
(c) type (6 *3)
(d) type (6 / 3)
(e) type (6 // 3)
(f) type (6 % 3)

sol:
(a) type(6 + 3) = int
(b) type(6 - 3) = int
(c) type(6 * 3) = int
(d) type(6 / 3) = float
(e) type(6 // 3) = int
(f) type(6 % 3) = int

17) What are augmented assignment operators? How are they useful?

sol:
- Augmented assignment operators are special operators that combine an arithmetic operation with assignment.
- They are useful because they allow you to update the value of a variable in a shorter and simpler way.
- Examples: +=, -=, *=, /=, //=, %=, **=

18) Differentiate between (555/222)**2 and (555.0/222)**2.

sol:

Expression	Type of Operands	Result Type	Notes
(555/222)**2	int / int	float	Division of integers produces a float
(555.0/222)**2	float / int	float	Using float ensures decimal precision

19) Given three Boolean variables a, b, c as : a = False, b = True, c = False. Evaluate the following Boolean expressions:

(a) b and c
(b) b or c
(c) not a and b
(d) (a and b) or not c
(e) not b and not (a or c)
(f) not ((not b or not a) and c) or a

sol:
(a) b and c → False
(b) b or c → True
(c) not a and b → True
(d) (a and b) or not c → True
(e) not b and not (a or c) → False
(f) not ((not b or not a) and c) or a → True

20) What would following code fragments result in? Given x = 3.

(a) 1 < x
(b) x >= 4
(c) x == 3
(d) x == 3.0
(e) "Hello" == "Hello"
(f) "Hello" > "hello"
(g) 4/2 == 2.0
(h) 4/2 == 2
(i) x < 7 and 4 > 5.

sol:
(a) 1 < x → True
(b) x >= 4 → False
(c) x == 3 → True
(d) x == 3.0 → True
(e) "Hello" == "Hello" → True
(f) "Hello" > "hello" → False
(g) 4/2 == 2.0 → True
(h) 4/2 == 2 → True
(i) x < 7 and 4 > 5 → False

21) it is for next lesson (out of lesson)

22) int('a') produces error. Why ?

sol:
- int('a') produces an error because 'a' is not a numeric value.
- The int() function can only convert strings that represent numbers, like '10' or '3', into integers.
- Since 'a' is a letter, Python cannot convert it to an integer, so it raises a ValueError.

23) int('a') produces error but following expression having int('a') in it, does not return error. Why?
len('a') + 2 or int('a')

sol:

Why len('a') + 2 or int('a') Does Not Give an Error

In the expression len('a') + 2 or int('a'):

- len('a') + 2 is evaluated first. It gives 1 + 2 = 3, which is True in a Boolean context.
- In an or operation, Python stops evaluating as soon as it finds a True value (short-circuiting).
- Since the first part is True, int('a') is never executed, so no error occurs.

This is called short-circuit evaluation.

24) Write expression to convert the values 17, len('ab') to (i) integer (ii) str (iii) boolean values

sol:

(i) Convert to integer:

```
int(17)         # 17
int(len('ab'))    # 2
```

(ii) Convert to string:

```
str(17)         # "17"
str(len('ab'))    # "2"
```

(iii) Convert to boolean:

```
bool(17)         # True
bool(len('ab'))   # True
```

25) Evaluate and Justify:

(i) 22 / 17 = 37 / 47 + 88 /83
(ii) len('375')**2

sol:

(i) 22 / 17 = 37 / 47 + 88 / 83

- Left-hand side: 22 / 17 ≈ 1.2941
- Right-hand side: 37 / 47 + 88 / 83 ≈ 0.7872 + 1.0602 ≈ 1.8474
- Comparison: 1.2941 == 1.8474 → False

Justification: The two sides are not equal because the sum of the fractions on the right is greater than the fraction on the left.

(ii) len('375')**2

- len('375') = 3 (there are 3 characters in the string '375')
- Squaring it: 3 ** 2 = 9

26) Evaluate and Justify:

(i) 22.0/7.0 - 22/7
(ii) 22.0/7.0 - int(22.0/7.0)

(iii) 22/7 - int (22.0)/7

sol:

(i) 22.0/7.0 - 22/7

- 22.0/7.0 ≈ 3.142857142857143 (float division)
- 22/7 = 3.142857142857143 (Python 3 division always gives float)
- Difference: 3.142857142857143 - 3.142857142857143 = 0.0
- Result: 0.0

Justification: Both divisions give the same float value, so their difference is 0.

(ii) 22.0/7.0 - int(22.0/7.0)

- 22.0/7.0 ≈ 3.142857142857143
- int(22.0/7.0) = int(3.142857142857143) = 3
- Difference: 3.142857142857143 - 3 = 0.142857142857143
- Result: ≈ 0.142857

Justification: Subtracting the integer part from the float gives the decimal (fractional) part.

(iii) 22/7 - int(22.0)/7

- 22/7 ≈ 3.142857142857143
- int(22.0)/7 = 22/7 = 3.142857142857143
- Difference: 3.142857142857143 - 3.142857142857143 = 0.0
- Result: 0.0

Justification: int(22.0) is 22, so dividing by 7 gives the same value as 22/7. Difference is 0.

27) Evaluate and Justify:

(i) false and None
(ii) 0 and None
(iii) True and None
(iv) None and None

sol:

(i) False and None

- and returns the first **False** value it finds.
- Result: **False**

(ii) 0 and None

- 0 is treated as False in Python.
- and returns the first False value.
- Result: **0**

(iii) True and None

- and returns the first False value, or the last value if all are True.
- True is True, next is None → Result: **None**

(iv) None and None

- None is treated as False.
- and returns the first False value → Result: **None**

Justification: In Python, the and operator returns the first value that is **False** in a Boolean context; if all are True, it returns the last value.

28) Evaluate and Justify:

(a) 0 or None and "or"
(b) 1 or None and 'a' or 'b'
(c) False and 23
(d) 23 and False
(e) not (1 == 1 and 0 != 1)
(f) "abc" == "Abc" and not (2 == 3 or 3 == 4)
(g) False and 1 == 1 or not True or 1 == 1 and False or 0 == 0

sol:

(a) 0 or None and "or"

- None and "or" → None (first False value)
- 0 or None → None (first True/False logic in or)
- **Result:** None

(b) 1 or None and 'a' or 'b'

- None and 'a' → None
- 1 or None → 1 (first True value)
- 1 or 'b' → 1
- **Result:** 1

(c) False and 23

- and returns first False → False
- **Result:** False

(d) 23 and False

- and returns first False → False
- **Result:** False

(e) not (1 == 1 and 0 != 1)

- 1 == 1 → True
- 0 != 1 → True
- True and True → True
- not True → False
- **Result:** False

(f) "abc" == "Abc" and not (2 == 3 or 3 == 4)

- "abc" == "Abc" → False
- 2 == 3 or 3 == 4 → False or False → False
- not False → True
- False and True → False
- **Result:** False

(g) False and 1 == 1 or not True or 1 == 1 and False or 0 == 0

Step by step:

1. False and 1 == 1 → False
2. not True → False
3. 1 == 1 and False → True and False → False
4. 0 == 0 → True

Combine with or: False or False or False or True → True

- Result: True

29) Evaluate the following for each expression that is successfully evaluated, determine its value and type for unsuccessful expression, state the reason.

(a) len("hello") == 25/5 or 20/10
(b) 3 < 5 or 50/(5 - (3 + 2))
(c) 50/(5 - (3 + 2)) or 3 < 5
(d) 2 * (2 * (len("01")))

sol:

(a) len("hello") == 25/5 or 20/10

- len("hello") → 5
- 25 / 5 → 5.0
- 5 == 5.0 → True
- True or 20/10 → True (short-circuit, second part not evaluated)
- Value: True
- Type: bool

(b) 3 < 5 or 50/(5 - (3 + 2))

- 3 < 5 → True
- True or ... → True (short-circuit, second part not evaluated)
- Value: True
- Type: bool

(c) 50/(5 - (3 + 2)) or 3 < 5

- 5 - (3 + 2) → 0
- 50 / 0 → Division by zero error
- Result: Error
- Reason: Division by zero

(d) 2 * (2 * (len("01")))

- len("01") → 2
- 2 * 2 * 2 → 8
- Value: 8
- Type: int

30) Write an expression that uses exactly 3 arithmetic operators with integer literals and produces result as 99.

sol:

50 + 25 * 2 - 1

31) Add parentheses to the following expression to make the order of evaluation more clear.
y % 4 == 0 and y % 100 != 0 or y % 400 == 0

sol:
((y % 4 == 0) and (y % 100 != 0)) or (y % 400 == 0)

32) A program runs to completion but gives an incorrect result. What type of error would have caused it?

sol:

- A program that runs to completion but gives an incorrect result has a logical error.
- Logical errors happen when the code is correct syntactically, but the logic used is wrong, so the output is not as expected.

33) In Python, strings are immutable while lists are mutable. What is the difference?

sol:

Feature	Strings (Immutable)	Lists (Mutable)
Changeable	Cannot be changed after creation	Can be changed after creation
Operations	Can create new strings only	Can modify, add, or remove items
Example	"Hello" → cannot change 'H'	[1, 2, 3] → can change 1 to 5

34) How does the // operator differ from the / operator? Give an example of where // would be needed.

sol:

Operator	Meaning	Result Example	Use Case Example
/	Division (normal)	7 / 2 = 3.5	When you need exact decimal result
//	Floor Division (integer)	7 // 2 = 3	When you need only the whole number (e.g., full boxes)

35) MidAir Airlines will only allow carry-on bags that are no more than 22 inches long, 14 inches wide, and 9 inches deep. Assuming that variables named length, width, and depth have already been assigned values, write an expression combining the three that evaluates to True if bag fits within those limits, and False otherwise.

sol:
length <= 22 and width <= 14 and depth <= 9

36) What are main error types? Which types are most dangerous and why?

sol:
Main Error Types in Python

Syntax Error – Mistakes in the structure of the code, like missing a colon or parenthesis.
Runtime Error – Errors that occur while the program is running, like division by zero.
Logical Error – The program runs without crashing but gives the wrong result due to incorrect logic.

Most dangerous: Logical errors, because the program completes without stopping, but the output is wrong, which can go unnoticed and cause bigger problems.

37) Correct any false statements:

(a) Compile-time errors are usually easier to detect and to correct than run-time errors.
(b) Logically errors can usually be detected by the compiler.

sol:

(a) Compile-time errors are usually easier to detect and correct than run-time errors. This is true.

(b) Logical errors can usually be detected by the compiler. This is false. Logical errors cannot be detected by the compiler; they are detected by testing and debugging the program.

38) Differentiate between a syntax error and a semantics error.

sol:

Feature	**Syntax Error**	**Semantics Error**
Meaning	Mistake in the structure or format of code	Mistake in the logic of the code
Detected by	Compiler or interpreter	Only detected when program runs
Effect	Program will not run	Program runs but gives wrong result
Example	Missing colon in if statement	Calculating area of rectangle as length + width instead of length * width

39) Differentiate between a syntax error and a logical error in a python program. When is each type of error likely to be found?

sol:

Feature	**Syntax Error**	**Logical Error**
Meaning	Mistake in the structure or format of code	Mistake in the logic of the program
Detected by	Python interpreter before running the code	Only detected when the program runs
Effect	Program will not run	Program runs but gives wrong result
When Found	Found while writing or compiling the code	Found during testing or after running the program

40) What is the difference between an error and exception?

sol:

Feature	Error	Exception
Meaning	A serious problem in the program that usually cannot be handled	An event that occurs during program execution and can be handled using code
Effect	Stops the program immediately	Can be caught and handled, program may continue
Example	Syntax error, memory error	ZeroDivisionError, FileNotFoundError