1. Python Basics – Answers

1. Types of comments in Python

- **Single-line comment:** Uses #

```
# This is a single-line comment
```

- **Multi-line comment:** Uses triple quotes (`''' '''` or `""" """`)

```
"""
This is a
multi-line comment
"""
```

---

2. Dynamic typing

Python determines the **data type at runtime**, not during compilation.
You can assign any type of value to a variable without declaring its type.

```
x = 10      # x is int
x = "Hello" # now x is string
```

---

3. Mutable vs Immutable data types

- **Mutable (can be changed):** list, dict, set, bytearray
- **Immutable (cannot be changed):** int, float, string, tuple, bool, bytes

---

4. Type casting

Converts one data type to another:

- `int("10")` → `10`
- `float("10.5")` → `10.5`
- `str(100)` → `"100"`

---

5. Python identifiers & naming rules

Identifiers = names of variables, functions, classes, etc.
Rules:

- Cannot start with a number
- Cannot use special characters (`!@#%`)
- Case-sensitive
- Use _ for spaces

Convention: `snake_case` for variables, `PascalCase` for classes.

---

6. Variable inside vs outside function

- **Inside function:** Local variable (scope limited to function)
- **Outside function:** Global variable
  Example:

```
x = 10   # global

def f():
    x = 5   # local
```

---

7. `input()` function

Used to take input as **string**.

```
n = input("Enter: ")
```

Convert to integer:

```
n = int(input("Enter number: "))
```

---

2. Strings – Answers

8. Storage & mutability

- Strings are stored as sequences of characters.
- **Strings are immutable** → cannot be changed after creation.

---

9. String slicing

Extract a substring using `[start:end:step]`.

```
s = "Python"
s[0:3]   # 'Pyt'
s[::-1]  # reverse
```

---

10. strip(), lstrip(), rstrip()

- `strip()` → removes leading + trailing spaces
- `lstrip()` → removes left spaces
- `rstrip()` → removes right spaces

---

11. find(), index(), count(), replace()

- `find()` → returns index or -1
- `index()` → returns index or raises error
- `count()` → number of occurrences
- `replace(a, b)` → replaces substring

---

3. Data Collections – Answers

12. List vs Tuple

List	Tuple
Mutable	Immutable
Slower	Faster

List	Tuple
Uses `[]`	Uses `()`

---

13. Add, update, delete from list

```
lst.append(10)
lst[0] = 20       # update
del lst[1]        # delete
lst.remove(10)
```

---

14. List comprehension

Compact list creation:

```
squares = [x*x for x in range(5)]
```

---

15. What is a set?

- Unordered collection of unique items
- No duplicates
- Written as {}

---

16. Why are sets faster for membership?

Sets use **hashing**, giving O(1) average lookup time.

---

17. Dictionary keys/values & key rules

- Keys must be **immutable** (int, string, tuple)
- **List cannot be dict key** because it's mutable.

---

18. append() vs extend()

```
lst.append([1,2])  # adds as single element
```

```
lst.extend([1,2])  # adds elements individually
```

---

19. Merge dictionaries

Python 3.9+:

```
d = d1 | d2
```

Or:

```
d1.update(d2)
```

---

4. Operators – Answers

20. == vs is

- `==` → compares values
- `is` → compares memory identity

---

21. Logical operators

- `and`, `or`, `not`
  Example:

```
a > 10 and b < 20
```

---

22. Membership operators

- `in` → checks presence
- `not in` → checks absence

```
"a" in "apple"  # True
```

---

5. Conditional Statements – Answers

23. if/elif/else example

```
if x > 0:
    print("Positive")
elif x == 0:
    print("Zero")
else:
    print("Negative")
```

---

24. Nested if

`if` inside another `if`.

---

25. Ternary operator

```
result = "Even" if n % 2 == 0 else "Odd"
```

---

6. Loops – Answers

26. for vs while

- `for` → known iterations
- `while` → until condition false

---

27. Iterate list

```
for i in lst:
    print(i)
```

---

28. range()

Generates sequences:

```
range(5)        # 0-4
```

```
range(1,5)      # 1-4
range(1,10,2)   # step
```

---

29. else in loops

Runs **when loop completes normally** (no break):

```
for i in range(3):
    print(i)
else:
    print("Done")
```

---

7. Control Statements – Answers

30. break vs continue

- `break` → exits loop
- `continue` → skips current iteration

---

31. pass

A null statement; used as placeholder.

---

32. assert

Ensures condition is true; otherwise throws error.

```
assert x > 0
```

---

8. Functions, Recursion, Lambda, Generator, Iterator – Answers

33. What is a function?

Reusable block of code:

```
def add(a, b): return a+b
```

**34. *args vs kwargs

- `*args` → variable number of positional arguments
- `**kwargs` → variable number of keyword arguments

---

35. Recursion

Function calling itself:

```
def factorial(n):
    return 1 if n==0 else n * factorial(n-1)
```

---

36. Lambda function

Anonymous function:

```
square = lambda x: x*x
```

---

37. Difference: map(), filter(), reduce()

- `map()` → applies function to each element
- `filter()` → filters elements
- `reduce()` → reduces to single value (needs `functools`)

---

38. Iterator

Object used to iterate:

```
it = iter([1,2,3])
next(it)
```

---

39. Generator & yield

Generator returns one value at a time using `yield`.

```
def gen():
    yield 1
    yield 2
```

`yield` pauses function; `return` exits.

---

40. Default arguments

Assigned when argument not passed:

```
def f(a, b=10): pass
```

---

9. Try–Exception – Answers

41. try-except

```
try:
    x = 1/0
except ZeroDivisionError:
    print("Error")
```

---

42. finally

Runs **always**, regardless of error.

---

43. Custom exception

```
raise ValueError("Invalid input")
```

---

44. Exception vs specific errors

- `Exception` = base class
- Specific errors = subclasses (TypeError, ValueError, etc.)

---

10. File Handling – Answers

45. File modes

- `r` → read
- `w` → write (overwrite)
- `a` → append
- `rb` / `wb` → binary modes

---

46. Why use with open()

Automatically closes the file.

---

47. read(), readline(), readlines()

- `read()` → whole file
- `readline()` → one line
- `readlines()` → list of lines

---

11. OOPs – Answers

48. Four pillars

1. **Encapsulation** – hiding data
2. **Abstraction** – exposing essential features
3. **Inheritance** – reusing parent class
4. **Polymorphism** – many forms (method overriding/overloading)

---

49. Class vs instance variable

- Class variable → shared by all objects
- Instance variable → unique to each object

---

50. Inheritance

```
class A:
    pass

class B(A):
    pass
```

---

51. Polymorphism

Same method behaves differently in different classes.

---

52. @staticmethod & @classmethod

- `@staticmethod` → no access to class/object
- `@classmethod` → access to class (`cls`)

---

12. Pandas – Answers

53. Main Pandas structures

- **Series** (1D)
- **DataFrame** (2D)

---

54. Read CSV

```
df = pd.read_csv("file.csv")
```

---

55. loc vs iloc

- `loc` → label-based
- `iloc` → index-based

---

56. Handle missing values

```
df.dropna()
df.fillna(0)
```

---

57. Merge, join, concatenate

```
pd.merge(df1, df2)
pd.concat([df1, df2])
df1.join(df2)
```

---

58. Filter data

```
df[df["age"] > 30]
df[df["city"].isin(["Delhi"])]
```

---

59. GroupBy aggregation

```
df.groupby("city")["salary"].mean()
```

---

60. Change data types

```
df["age"] = df["age"].astype(int)
```
