# Python Interview Questions & Answers

Python Interview Questions & Answers (Detailed)

1. Casting in Python interview questions

- Casting means converting a variable from one data type to another.
- Python provides built-in functions for casting:
  - int(): Converts a value to an integer.
  - float(): Converts a value to a floating-point number.
  - str(): Converts a value to a string.
  - bool(): Converts a value to a boolean (True/False).
- Example:

```
x = "123"
y = int(x)  # y is 123 (int)
```

- Casting is useful when input data is in string form but numeric operations are needed.

2. Difference between .py and .pyc files

- .py file:
  - Contains Python source code.
  - Human-readable and editable.
  - Run by the Python interpreter.
- .pyc file:
  - Contains compiled Python bytecode.
  - Generated automatically by Python when a .py file is run.
  - Stored in __pycache__ directory.
  - Not human-readable.
  - Helps in faster program startup since Python loads bytecode instead of recompiling.
- The .pyc files are platform-independent bytecode executed by Python's virtual machine.

3. Different data types in Python

- int: Integer numbers, e.g., 10, -3.
- float: Floating-point numbers, e.g., 3.14, -0.001.

- complex: Complex numbers with real and imaginary parts, e.g., 1+2j.
- bool: Boolean values, True or False.
- str: String of characters, e.g., "Hello".
- list: Ordered, mutable sequence of values, e.g., [1, 2, 3].
- tuple: Ordered, immutable sequence, e.g., (1, 2, 3).
- set: Unordered collection of unique values, e.g., {1, 2, 3}.
- dict: Key-value pairs, e.g., {"name": "Alice", "age": 25}.
- NoneType: Represents the absence of a value (None).

4. Is it possible to modify items in a dictionary?
- Yes, dictionaries are mutable data types.
- You can add, update, or delete key-value pairs.
- Example:

```
my_dict = {"name": "Alice", "age": 25}
my_dict["age"] = 26  # Update age
my_dict["city"] = "NY"  # Add new key-value pair
```

5. How to access keys in a dictionary? Is there any method?
- Use dict.keys() method which returns a view object of keys.
- Example:

```
my_dict = {"name": "Alice", "age": 25}
keys = my_dict.keys()
for key in keys:
    print(key)
```

- You can also convert keys to a list: list(my_dict.keys())

6. Namespace in Python
- A namespace is a container that holds a set of identifiers (names) and their corresponding objects.
- Different namespaces exist:
  - Local Namespace: Inside a function.
  - Enclosing Namespace: In nested functions.

  - Global Namespace: At the module level.
  - Built-in Namespace: Contains built-in functions and exceptions.
- Namespaces prevent naming conflicts by isolating names in different contexts.

7. Variable scopes in Python
- The LEGB rule determines scope:
  - Local (L): Names defined inside a function.
  - Enclosing (E): Names in enclosing functions (nested functions).
  - Global (G): Names defined at the module level.
  - Built-in (B): Names pre-defined in Python.
- Python searches variables in this order when you reference a name.

8. Define a class 'Fruit' and create a method to get total cost for quantity
Example:

```
class Fruit:
    def __init__(self, name, price):
        self.name = name
        self.price = price
    def total_cost(self, quantity):
        return self.price * quantity
# Usage
apple = Fruit("Apple", 5)
print(apple.total_cost(10))  # Output: 50
```

9. What is the output of list slicing lst[-3:] where lst=[1,2,3,4,5]
- The slice lst[-3:] takes the last 3 elements of the list.
- So, output is: [3, 4, 5]

10. Python string replacements
- Use str.replace(old, new, count) to replace occurrences of a substring.
- Example:

```
text = "apple apple apple"
new_text = text.replace("apple", "orange", 2)
# Output: "orange orange apple"
```

11. Check if a string ends with any special character

- Use regular expressions (regex).
- Pattern to check last character is special (non-alphanumeric): r'[^a-zA-Z0-9]$'
- Example:

```
import re
def ends_with_special(s):
    return bool(re.search(r'[^a-zA-Z0-9]$', s))
```

12. Working with dates in Python

- Dates are not a native data type.
- Use 'datetime' module for working with dates and times.
- Commonly used classes: datetime.date, datetime.datetime, datetime.timedelta.
- Examples:

```
import datetime
today = datetime.date.today()
now = datetime.datetime.now()
tomorrow = today + datetime.timedelta(days=1)
```

13. Print only the keys from a dictionary

- Use dict.keys() method.
- Example:

```
my_dict = {"a":1, "b":2}
for key in my_dict.keys():
    print(key)
```

14. Functions in Python - different types

- Built-in functions: Provided by Python.

- User-defined functions: Defined by user using def keyword.
- Lambda (anonymous) functions: Small, unnamed functions using lambda.
- Recursive functions: Functions that call themselves.
- Generator functions: Use yield to produce sequence lazily.
- Higher-order functions: Accept other functions as arguments or return functions.
- Decorators: Functions modifying other functions' behavior.

15. Difference between list and array in Python

- List:
  - Built-in type, flexible.
  - Can hold different data types.
  - Slower and uses more memory.
- Array:
  - From array module.
  - Holds elements of the same type.
  - More efficient for numeric data.
  - Limited functionality compared to lists.

16. Use of 'pass' statement

- `pass` is a placeholder that does nothing.
- Used to create empty code blocks without syntax errors.
- Example:

```
def func():
    pass  # To be implemented later
```