Module-5, :- Theory

-> Python Fundamentals Introduction to Python Theory :-

Q1. Introduction to Python and its Features ?

Ans:- Python is a **simple**, **high-level**, and **interpreted** programming language used for web development, data science, automation, and more.
Features:

- Easy to read and write
- Interpreted (no need to compile)
- Dynamically typed
- Large standard library
- Platform-independent

Q2. History and Evolution of Python ?

Ans:- Developed by **Guido van Rossum** in **1989**, released in **1991**

- Named after the TV show *"Monty Python's Flying Circus"*
- **Python 2** was popular but later replaced
- **Python 3** was released in **2008** with many improvements

Q3. Advantages of using Python over other programming languages ?

Ans:-

- Si
- .00mple syntax (like English)
- Huge community support
- Lots of libraries (NumPy, Pandas, etc.)
- Great for beginners
- Works for web, AI, data science, etc.

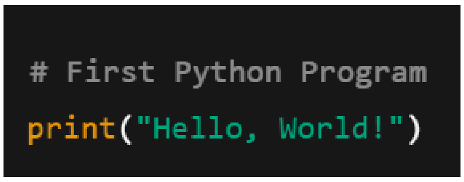
Q4. Installing Python and Setting Up Development Environment ?

Ans:- **Ways to install and use Python:**

- **Official website:** Install from python.org
- **Anaconda:** Best for data science and Jupyter notebooks
- **PyCharm or VS Code:** Popular IDEs for writing Python code

Q5. Writing and Executing Your First Python Program ?

Ans:-



```python
# First Python Program
print("Hello, World!")
```

To run:

- In command line: `python file_name.py`
- Or use an IDE like VS Code or Jupyter Notebook

Q6. Understanding Python's PEP 8 Guidelines ?

Ans:- **PEP 8** is Python's style guide.
It recommends:

- Use 4 spaces for indentation
- Use lowercase_with_underscores for variable names
- Keep code clean and readable

Q7. Indentation, Comments, and Naming Conventions in Python ?

Ans:- **Indentation** is mandatory in Python (no `{}` or `end`)

- **Comments:**
  - Single-line: `# This is a comment`
  - Multi-line: `'''comment'''` or `"""comment"""`
- **Naming:**
  - Variables: `user_name, total_amount`
  - Constants: `PI = 3.14`

    - Classes: class MyClass:

Q8. Writing Readable and Maintainable Code ?

Ans:- Tips:

- Use meaningful variable names
- Add comments where needed
- Follow PEP 8
- Keep functions small and focused
- Avoid duplicate code

Q9. Understanding Data Types ?

Ans:- **int**: Whole numbers → `x = 10`

- **float**: Decimal numbers → `pi = 3.14`
- **str**: Text → `name = "Alice"`
- **list**: Ordered, changeable → `fruits = ['apple', 'banana']`
- **tuple**: Ordered, unchangeable → `coords = (1, 2)`
- **dict**: Key-value pairs → `student = {'name': 'Bob', 'age': 20}`
- **set**: Unordered, no duplicates → `unique = {1, 2, 3}`

Q10. Python Variables and Memory Allocation ?

Ans:-

- Variables are created when assigned → `a = 5`
- Python handles memory automatically using **dynamic memory management**
- Each object has a unique **ID** in memory

Q11. Python Operators ?

Ans:-

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

Q12. Introduction to Conditional Statements: if, else, elif ?

Ans:- Conditional statements let you run code based on conditions:

```python
x = 10
if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is 5")
else:
    print("x is less than 5")
```

Q13. Nested if-else Conditions ?

Ans:- You can place if-else inside another if-else for multiple checks:

```python
x = 15
if x > 10:
    if x < 20:
        print("x is between 10 and 20")
    else:
        print("x is 20 or more")
else:
    print("x is 10 or less")
```

Q14. Introduction to for and while Loops ?

Ans:- for loop**:** Iterates over items

- **`while`** loop: Runs while a condition is true

```python
# for loop
for i in range(5):
    print(i)

# while loop
x = 0
while x < 5:
    print(x)
    x += 1
```

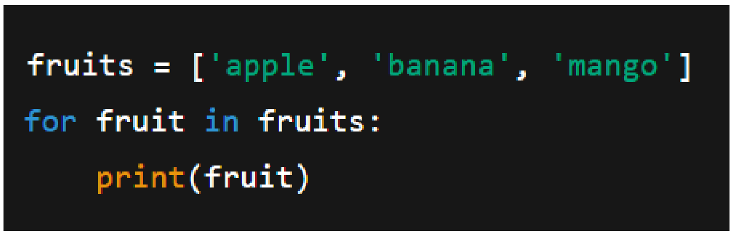
Q15. How Loops Work in Python ?

Ans:-

- Loops **repeat a block** of code
- `break` exits the loop early
- `continue` skips to next iteration
- `else` runs after loop ends (if not broken)

Q16. Using Loops with Collections ?

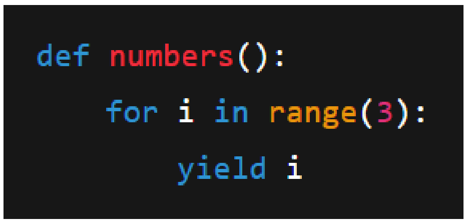
Ans:- You can loop over lists, tuples, strings, etc.



```python
fruits = ['apple', 'banana', 'mango']
for fruit in fruits:
    print(fruit)
```

Q17. Understanding How Generators Work in Python ?

Ans:- Generators generate values one by one using `yield`.
They are memory efficient for large data.

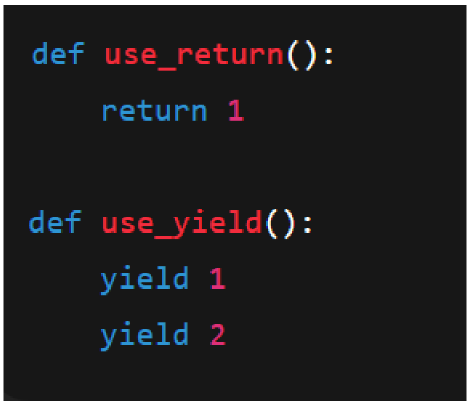


```python
def numbers():
    for i in range(3):
        yield i
```

Q18. Difference Between yield and return ?

Ans:-

- `return` ends the function and gives one value
- `yield` pauses the function and returns a **generator**



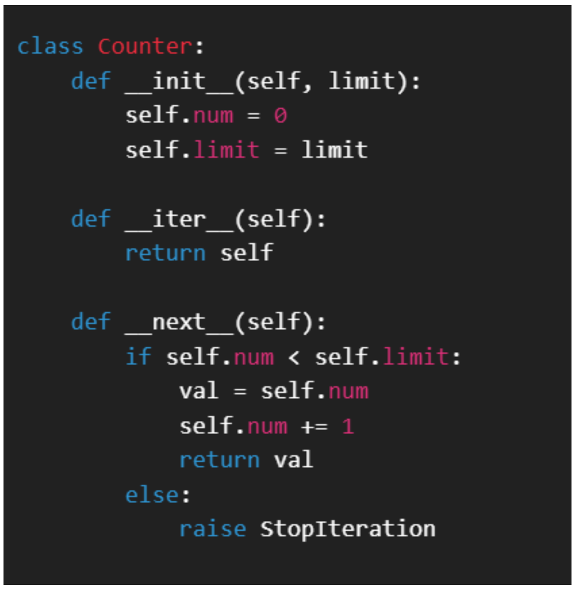
```python
def use_return():
    return 1

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

Q19. Understanding Iterators and Custom Iterators ?

Ans:-

- **Iterator**: Object you can loop over using `next()`
- You can create custom iterators using classes:



```python
class Counter:
    def __init__(self, limit):
        self.num = 0
        self.limit = limit

    def __iter__(self):
        return self

    def __next__(self):
        if self.num < self.limit:
            val = self.num
            self.num += 1
            return val
        else:
            raise StopIteration
```

Q20. Defining and Calling Functions in Python ?

Ans:-   Functions help you reuse code**:**

```
def greet(name):
    print("Hello", name)

greet("Swastik")
```

Q22. Function Arguments: Positional, Keyword, Default ?

Ans:-

```
def info(name, age=18):   # age has default value
    print(name, age)

info("Ravi")                    # Positional
info(name="Ravi", age=20)       # Keyword
```

Q23. Scope of Variables in Python ?

Ans:-

- **Local**: Inside function
- **Global**: Outside function

```
x = 10  # global

def show():
    x = 5  # local
    print(x)
```

Q24. Built-in Methods for Strings, Lists, etc. ?

Ans:- String methods:

```
s = "hello"
s.upper()      # 'HELLO'
s.replace("h", "y")    # 'yello'
```

Q25. Understanding the Role of break, continue, and `pass` in Python Loops ?

Ans:-

- **break**: Exits the loop early.
- **continue**: Skips the current iteration and moves to the next.
- **pass**: Does nothing — used as a placeholder.

```
for i in range(5):
    if i == 3:
        break    # stops loop at 3
    print(i)

for i in range(5):
    if i == 2:
        continue   # skips 2
    print(i)

for i in range(3):
    pass   # does nothing
```

Q26. Understanding How to Access and Manipulate Strings ?

Ans:- Strings are sequences of characters and can be accessed using indexing:

```
text = "hello"
print(text[0])   # 'h'
print(text[-1])  # 'o'
```

You can also modify or combine them:

```
new = text + " world"  # Concatenation
```

Q27. Basic String Operations ?

Ans:- Concatenation: `s1 + s2`

- Repetition: s * 3
- String methods:

```
s = "Python"
s.upper()     # 'PYTHON'
s.lower()     # 'python'
s.replace('P', 'J')  # 'Jython'
s.strip()     # removes spaces
s.find('t')   # finds index of 't'
```

Q28. String Slicing ?

Ans:- Slicing lets you get a part of the string:

```
text = "Hello, World"
print(text[0:5])    # 'Hello'
print(text[:5])     # 'Hello'
print(text[7:])     # 'World'
print(text[-5:])    # 'World'
```

Q29. How Functional Programming Works in Python ?

Ans:- Functional programming focuses on:

- Using pure functions (no side effects)
- Using functions as arguments
- Avoiding changes to variables

Python supports functional programming with:

- `map()`, `filter()`, `reduce()`
- `lambda` functions
- Higher-order functions

Q30. Using map(), reduce(), and filter() Functions ?

Ans:-

- **`map()`**: Applies a function to each item

```
nums = [1, 2, 3]
squares = list(map(lambda x: x*x, nums))  # [1, 4, 9]
```

`filter()`: Filters items based on condition

```
even = list(filter(lambda x: x%2==0, nums))  # [2]
```

`reduce()` (from functools): Reduces to a single value

```
from functools import reduce
product = reduce(lambda x, y: x * y, nums)  # 6
```

Q31. Introduction to Closures and Decorators ?

Ans:-

- **Closure**: A function defined inside another function that remembers the outer function's variables.

```
def outer(x):
    def inner():
        print(x)
    return inner

f = outer(10)
f()  # prints 10
```

Decorator: A function that modifies another function's behavior.

```
def decorator(func):
    def wrapper():
        print("Before function")
        func()
        print("After function")
    return wrapper

@decorator
def greet():
    print("Hello")

greet()
```