🐍 User-Defined Functions in Python - Interview Preparation

🎯 Objective:

Prepare for Python interviews with an interactive approach to **user-defined functions (UDFs)** by covering key concepts, hands-on examples, and commonly asked interview questions.

✅ 1. What is a User-Defined Function in Python?

A **User-Defined Function (UDF)** is a function created by the user to perform specific tasks.

💡 Syntax:

```
tdef function_name(parameters):
    """Docstring (optional): Describes the function's purpose."""
    # Code block
    return value  # (optional)
```

📚 2. Types of Functions in Python

1. **Without Parameters and Return Value**

```
    def greet():

        print("Hello, Welcome to Python!")

greet()
```

👉 *Output:*

```
Hello, Welcome to Python!
```

---

2. **With Parameters and Without Return Value**

```
def add(a, b):
    print("Sum:", a + b)

add(5, 3)
```

👉 *Output:*

```
Sum: 8
```

---

3. **With Parameters and Return Value**

```
def multiply(x, y):
    return x * y

result = multiply(4, 5)
print("Result:", result)
```

👉 *Output:*

```
Result: 20
```

---

4. **Without Parameters and With Return Value**

```
def get_pi():
    return 3.14159

value = get_pi()
print("Value of Pi:", value)
```

👉 *Output:*

```
Value of Pi: 3.14159
```

📝 3. Key Concepts to Understand

➢ 1. Arguments vs Parameters

- **Parameters -** Variables defined in the function header.
- **Arguments -** Values passed during a function call.

➢ 2. Positional and Keyword Arguments

- **Positional Argument:**

```
def info(name, age):
    print(f"{name} is {age} years old.")

info("John", 25)
```

- **Keyword Argument:**

python
CopyEdit
```
info(age=25, name="John")
```

➢ 3. Default Parameters

```
def greet(name="User"):
    print(f"Hello, {name}!")

greet()
greet("Sanket")
```

👉 *Output:*

```
Hello, User!

Hello, Sanket!
```

➢ 4. Arbitrary Arguments (`*args`)

```
def sum_all(*numbers):
    return sum(numbers)

print(sum_all(1, 2, 3, 4, 5))
```

👉 *Output:*

```
15
```

➢ 5. Keyword Arbitrary Arguments (`**kwargs`)

```
def display_info(**data):
    for key, value in data.items():
        print(f"{key}: {value}")

display_info(name="Sanket", age=30, city="Rajkot")
```

👉 *Output:*

```
name: Sanket
age: 30
city: Rajkot
```

🧠 4. Common Interview Questions on UDFs

🔥 1. What is the difference between `return` and `print` in a function?

- `return` sends a value back to the caller.
- `print` displays output but doesn't affect the function's return value.

🔥 2. Can we pass a function as an argument?

Yes!

```
def square(n):
    return n * n

def apply_func(func, value):
    return func(value)

result = apply_func(square, 5)
print(result)
```

👉 *Output:*

```
25
```

🔥 3. What is recursion in Python?

A function that calls itself.

Example:

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

print(factorial(5))
```

👉 *Output:*

```
120
```

🕹️ 5. Practice Challenges

1. **Reverse a String using a Function**

```
def reverse_string(s):
    return s[::-1]

print(reverse_string("Python"))
```

2. **Find Maximum of Three Numbers**

```
def find_max(a, b, c):
    return max(a, b, c)

print(find_max(10, 20, 15))
```

3. **Check Prime Number**

```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

print(is_prime(13))  # True
print(is_prime(8))   # False
```

🎯 6. Advanced Concepts to Cover

1. **Lambda Functions**

```
add = lambda x, y: x + y
print(add(3, 5))  # 8
```

2. **Nested Functions**

```
def outer_function():
    def inner_function():
        print("Inner function executed!")
    inner_function()

outer_function()
```

3. **Closure in Python**

```
def multiplier(n):
    def inner(x):
        return x * n
    return inner

double = multiplier(2)
print(double(5))  # 10
```

🎁 7. Quick Tips for Interview Success

- ✅ Understand function scope (local, global).
- ✅ Practice recursion, *args, and **kwargs.
- ✅ Be prepared to optimize functions for performance.

- ✅ Discuss time and space complexity when asked.
- ✅ Write clean and modular code.
