Python Notes – Function – Notes

Prepared by: **Dhananjay Kumar**
Trainer | Java Full-Stack & Python | Contact: 7023577968
LinkedIn: Dhananjay Kumar

Today's Agenda

- Introduction to Functions
- Why Functions?
- Different Types of Functions
- User Defined Functions in detail

Introduction to Functions

Definition: A Function is a set of statements that takes inputs, performs some specific task, and produces output.

Steps to declare a function in Python:

1. Use the keyword `def` to declare the function followed by the function name.
2. Add parameters to the function (inside parentheses). End the line with a colon `:`.
3. Add statements that the function should execute.
4. End with a `return` statement if the function should output something.

Example:

```
def greet():
    print("Hello, Welcome to Python Functions!")

greet()
```

Why Functions?

Functions are useful because:

- **Reusability**: Write once, use many times.
- **Modularity**: Divide large programs into smaller blocks.
- **Readability & Maintainability**

Without Functions:

```
a = 10
b = 20
c = a + b
print(c)

d = 100
e = 10
f = d / e
print(f)
```

If we need the same operations again, we must **repeat the code** → not efficient.

With Functions (DRY Principle – Don't Repeat Yourself):

```
def add(x, y):
    return x + y

def divide(x, y):
    return x / y

print(add(10, 20))
print(divide(100, 10))
```

Functions Behaviour

- When a function is called, **stack frames** are created for execution.
- Python automatically manages memory using the **garbage collector** once references are lost.

Types of Functions in Python

Python has 4 types of functions:

1. **User Defined Functions** → Created by the user.
2. **Built-in Functions** → Already available in Python (`len()`, `max()`, etc.).
3. **Lambda Functions** → Anonymous one-line functions using `lambda`.
4. **Recursive Functions** → A function that calls itself.

Example of Built-in Functions:

```
print(len("Python"))
print(max([3, 7, 2, 9]))
```

Example of Lambda Function:

```
add = lambda a, b: a + b
print(add(5, 10))
```

Example of Recursive Function:

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

print(factorial(5))  # 120
```

User Defined Functions

There are **4 categories** of user-defined functions:

1. Function with **No Input & No Output**

```
def mul():
    a = 10
    b = 20
    c = a * b
    print("Multiplication:", c)

# main function execution
mul()
```

---

2. Function with **Input & No Output**

```
def mul(x, y):
    print("Multiplication:", x * y)

mul(10, 20)
```

---

3. Function with **No Input & Returns Output**

```
def mul():
    a = 5
    b = 6
    return a * b

res = mul()
print("Returned Value:", res)
```

4. Function with **Input & Returns Output**

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

result = mul(7, 8)
print("Multiplication Result:", result)
```

Functions may return **multiple values** in Python:

```
def calc(x, y):
    return x+y, x-y, x*y, x/y

add, sub, mul, div = calc(20, 10)
print("Addition:", add)
print("Subtraction:", sub)
print("Multiplication:", mul)
print("Division:", div)
```

Summary

- Functions make programs modular, reusable, and maintainable.
- Python supports **4 types of user-defined functions**.
- Functions can return single or multiple values.
- Python memory management uses **stack frames** and a **garbage collector**.

Python Functions Practice Questions (30 with Solutions & Explanations)

Practice Set: Functions in Python

This notebook contains **30 practice problems** related to Python functions. Each problem includes a solution and an explanation.

Q1: Write a function to print "Hello Python!"

```
def say_hello():
    print("Hello Python!")

say_hello()
```

Explanation: This is a simple function with no input and no return value.

Q2: Write a function that takes a name as input and prints a greeting.

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

greet("Dhananjay")
```

Explanation: Function with input but no return value.

Q3: Write a function to add two numbers and return the result.

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

print(add(10, 20))
```

Explanation: Function with input and return.

Q4: Write a function that multiplies two numbers but only prints the result.

```
def multiply(a, b):
    print("Product:", a * b)

multiply(5, 6)
```

Explanation: Input with no return.

Q5: Write a function that returns the square of a number.

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

print(square(7))
```

Explanation: Demonstrates returning values.

Q6: Write a function to check whether a number is even or odd.

```
def even_odd(n):
    if n % 2 == 0:
        return "Even"
    else:
        return "Odd"

print(even_odd(7))
```

Explanation: Conditional logic inside functions.

Q7: Write a function that calculates factorial using recursion.

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

print(factorial(5))
```

Explanation: Recursive function.

Q8: Write a function that calculates factorial using iteration.

```
def factorial_iter(n):
    fact = 1
    for i in range(1, n+1):
        fact *= i
    return fact

print(factorial_iter(5))
```

Explanation: Iterative approach to recursion.

Q9: Write a function that takes a list of numbers and returns the sum.

```
def sum_list(numbers):
    return sum(numbers)

print(sum_list([1, 2, 3, 4, 5]))
```

Explanation: Passing lists as input.

Q10: Write a function to find the maximum of three numbers.

```python
def maximum(a, b, c):
    return max(a, b, c)

print(maximum(10, 20, 5))
```

Explanation: Using built-in function inside custom function.

Q11: Write a function that reverses a string.

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

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

Explanation: String slicing inside function.

Q12: Write a function that counts vowels in a string.

```python
def count_vowels(s):
    vowels = "aeiouAEIOU"
    return sum(1 for char in s if char in vowels)

print(count_vowels("Rooman Technologies"))
```

Explanation: Iterating through string.

Q13: Write a function that checks if a string is a palindrome.

```python
def is_palindrome(s):
    return s == s[::-1]

print(is_palindrome("madam"))
```

Explanation: Palindrome logic using slicing.

Q14: Write a function to calculate area of a circle given radius.

```python
def area_circle(r):
    return 3.14 * r * r

print(area_circle(7))
```

Explanation: Formula inside function.

Q15: Write a function to calculate simple interest.

```python
def simple_interest(p, r, t):
    return (p * r * t) / 100

print(simple_interest(1000, 5, 2))
```

Explanation: Example with multiple inputs.

Q16: Write a function that returns multiple values (sum, difference).

```python
def calc(a, b):
    return a+b, a-b

s, d = calc(20, 10)
print("Sum:", s)
print("Difference:", d)
```

Explanation: Returning multiple values.

Q17: Write a function with default argument.

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

greet("Ankita")
greet()
```

Explanation: Demonstrates default parameters.

Q18: Write a lambda function to find cube of a number.

```
cube = lambda x: x**3
print(cube(3))
```

Explanation: One-line anonymous function.

Q19: Write a function that takes variable number of arguments.

```
def add_all(*args):
    return sum(args)

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

Explanation: `*args` allows multiple inputs.

Q20: Write a function that accepts keyword arguments.

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

person_info(name="Dhananjay", age=28, city="Patna")
```

Explanation: `**kwargs` allows dictionary-style input.

Q21: Write a function to check prime numbers.

```
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(11))
```

Explanation: Prime check function.

Q22: Write a recursive function to calculate Fibonacci series.

```
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

print([fibonacci(i) for i in range(7)])
```

Explanation: Recursive Fibonacci.

Q23: Write an iterative function for Fibonacci series.

```
def fibonacci_iter(n):
    seq = [0, 1]
    for i in range(2, n):
```

```
        seq.append(seq[-1] + seq[-2])
    return seq[:n]

print(fibonacci_iter(7))
```
Explanation: Iterative Fibonacci.

Q24: Write a function that converts Celsius to Fahrenheit.

```
def c_to_f(c):
    return (c * 9/5) + 32

print(c_to_f(37))
```
Explanation: Conversion function.

Q25: Write a function that finds the minimum in a list.

```
def find_min(lst):
    return min(lst)

print(find_min([8, 2, 5, 1, 9]))
```
Explanation: Wrapper around built-in function.

Q26: Write a function to count words in a sentence.

```
def word_count(sentence):
    return len(sentence.split())

print(word_count("Python is fun to learn"))
```
Explanation: Using string split.

Q27: Write a function that returns square of all numbers in a list.

```
def square_list(lst):
    return [x**2 for x in lst]

print(square_list([1, 2, 3, 4]))
```
Explanation: List comprehension inside function.

Q28: Write a function that checks if a number is Armstrong.

```
def is_armstrong(n):
    digits = str(n)
    power = len(digits)
    return n == sum(int(d)**power for d in digits)

print(is_armstrong(153))
```
Explanation: Armstrong number check.

Q29: Write a function to swap two numbers.

```
def swap(a, b):
    return b, a

x, y = swap(10, 20)
print("After Swap:", x, y)
```
Explanation: Returning swapped values.

Q30: Write a function to generate multiplication table of a

number.

```
def multiplication_table(n):
    for i in range(1, 11):
        print(f"{n} x {i} = {n*i}")

multiplication_table(5)
```

Explanation: Demonstrates loops inside functions.

Python Functions – MCQs with Answers & Explanations (30 Questions)

Multiple Choice Questions on Functions in Python

Q1. Which keyword is used to define a function in Python?

- a) func
- b) def
- c) function
- d) define

Answer: b) def
Explanation: Functions in Python are defined using the `def` keyword.

Q2. What is the default return value of a function that doesn't have a return statement?

- a) 0
- b) None
- c) False
- d) Error

Answer: b) None
Explanation: If no `return` statement is specified, Python returns `None` by default.

Q3. Which of the following is a valid function name?

- a) 123func
- b) my-function
- c) my_function
- d) def

Answer: c) my_function
Explanation: Function names must start with a letter or underscore and cannot use special characters like `-`.

Q4. Which of the following is a built-in function in Python?

- a) print()
- b) input()
- c) len()
- d) All of the above

Answer: d) All of the above
Explanation: Python provides many built-in functions such as `print()`, `input()`, and `len()`.

Q5. What does the following code return?

```
def test():
    return "Python"
print(test())
```

- a) None
- b) Error
- c) Python
- d) test

Answer: c) Python
Explanation: Function returns the string "Python".

Q6. Which type of function does not have a name?

- a) Recursive Function
- b) Lambda Function
- c) User-defined Function
- d) Built-in Function

Answer: b) Lambda Function
Explanation: Lambda functions are anonymous (without name).

Q7. Which statement is used to exit a function and return a value?

- a) exit
- b) break
- c) return
- d) stop

Answer: c) return
Explanation: The `return` statement exits the function and sends back a value.

Q8. What will be the output?

```
def add(x, y=5):
    return x + y
print(add(10))
```

- a) 5
- b) 10
- c) 15
- d) Error

Answer: c) 15
Explanation: Default argument `y=5` is used when not provided.

Q9. Which of the following can return multiple values in Python?

- a) return a, b
- b) return [a, b]
- c) return {a, b}
- d) All of the above

Answer: d) All of the above
Explanation: Python functions can return tuples, lists, sets, or any object.

Q10. What is the scope of a variable defined inside a function?

- a) Global
- b) Local
- c) Static
- d) Universal

Answer: b) Local
Explanation: Variables inside functions are local by default.

Q11. Which function calls itself in Python?

- a) Recursive
- b) Built-in
- c) User-defined
- d) Lambda

Answer: a) Recursive
Explanation: Recursive functions call themselves until a base condition is met.

Q12. What will be the output?

```
def myfunc():
    x = 10
print(x)
```

- a) 10
- b) None
- c) Error
- d) 0

Answer: c) Error
Explanation: Variable `x` is local to the function and cannot be accessed outside.

Q13. Which function converts a string into an integer?

- a) str()
- b) int()
- c) float()
- d) eval()

Answer: b) int()
Explanation: The `int()` function converts string/float into integer.

Q14. What does `*args` mean in function definition?

- a) It allows multiple keyword arguments
- b) It allows multiple positional arguments
- c) It defines global variables
- d) It is used for recursion

Answer: b) It allows multiple positional arguments
Explanation: `*args` collects multiple inputs into a tuple.

Q15. What does `**kwargs` mean in Python?

- a) Multiple keyword arguments

- b) Default arguments
- c) Arbitrary positional arguments
- d) Global variables

Answer: a) Multiple keyword arguments
Explanation: `**kwargs` collects keyword arguments into a dictionary.

Q16. What will be the output?

```
def f1():
    return 10
    print("Hello")
print(f1())
```

- a) 10
- b) Hello
- c) 10 and Hello
- d) Error

Answer: a) 10
Explanation: Code after `return` is never executed.

Q17. Which of these is NOT a function type in Python?

- a) Built-in
- b) User-defined
- c) Recursive
- d) Sequential

Answer: d) Sequential
Explanation: No such type exists.

Q18. Which of the following creates an anonymous function?

- a) def
- b) lambda
- c) return
- d) func

Answer: b) lambda
Explanation: `lambda` keyword creates anonymous functions.

Q19. What will be the output?

```
def f(x=[]):
    x.append(1)
    return x
print(f())
print(f())
```

- a) [1], [1]
- b) [1], [1, 1]
- c) Error
- d) None

Answer: b) [1], [1, 1]
Explanation: Default mutable arguments persist between calls.

Q20. Which keyword is used to access variables outside the current function?

- a) this
- b) outer
- c) global
- d) static

Answer: c) global
Explanation: `global` keyword allows modification of global variables.

Q21. What will be the output?

```
def f(a, b, c=5):
    return a+b+c
print(f(2, 3))
```

- a) 5
- b) 10
- c) 0
- d) Error

Answer: b) 10
Explanation: Default value `c=5` is used.

Q22. Which of these functions cannot be recursive?

- a) Fibonacci
- b) Factorial
- c) print()
- d) All can be recursive

Answer: c) print()
Explanation: Built-in functions like `print()` cannot be recursive.

Q23. What will be the output?

```
def f():
    x = 5
    return lambda y: x + y
val = f()
print(val(10))
```

- a) 5
- b) 10
- c) 15
- d) Error

Answer: c) 15
Explanation: Closure captures `x=5` and adds to `y=10` .

Q24. Which of the following is used for documentation of functions?

- a) Comments
- b) Docstring
- c) Notes
- d) Print statements

Answer: b) Docstring
Explanation: Triple quotes inside a function define docstring.

Q25. Which function is called first in a Python program?

- a) init()
- b) main()
- c) print()
- d) None

Answer: d) None
Explanation: Unlike C/C++, Python does not have a mandatory main(). Execution starts from top.

Q26. Which of these will create an error?

- a) def func(x, y): return x+y
- b) def func(x, y=10): return x+y
- c) def func(x=10, y): return x+y
- d) def func(): return 0

Answer: c) def func(x=10, y): return x+y
Explanation: Default arguments must come after non-default arguments.

Q27. What will be the output?

```
def f(a, b):
    return a**b
print(f(2, 3))
```

- a) 6
- b) 8
- c) 9
- d) 5

Answer: b) 8
Explanation: `2**3 = 8`.

Q28. Which of these allows infinite recursion in Python?

- a) Proper base condition
- b) Missing base condition
- c) return 0
- d) exit()

Answer: b) Missing base condition
Explanation: Without base condition, recursion continues until stack overflow.

Q29. Which built-in function returns the memory address of an object?

- a) id()
- b) type()
- c) address()
- d) ref()

Answer: a) id()
Explanation: `id()` returns unique memory location.

Q30. Which of these statements is true about Python functions?

- a) Functions must always return a value.
- b) Functions cannot be nested.
- c) Functions are first-class objects.

- d) Functions cannot be passed as arguments.

Answer: c) Functions are first-class objects.
Explanation: Functions can be assigned, passed as arguments, and returned.

In [ ]: