



Output: 300 300

Global Keyword

If you need to create a global variable, but are stuck in the local scope, you can use the global keyword.

```
def myfunc(): global x x = 300 myfunc() print(x)
```

Output: 300

□ Interview Question: What is the difference between local and global scope?

6. Recursion Introduction

Python also accepts function recursion, which means a defined function can call itself. Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

```
def tri_recursion(k): if(k > 0): result = k + tri_recursion(k - 1)
print(result) else: result = 0 return result print("Recursion Example
Results") tri_recursion(6)
```

Output: Recursion Example Results 1 3 6 10 15 21

□ Tip: The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power.

Common Mistakes (Recursion)

1. Missing Base Case: A recursive function must have a base case to stop calling itself. Without it, the function will run infinitely (or until it hits the recursion limit). 2. Not making progress towards the base case: The recursive call must change the state in a way that moves it closer to the base case.



UNIT 10 – FUNCTIONS

1. Function Basics

Functions are blocks of organized, reusable code that perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

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

Output: Hello, Alice!

Why use functions?

1. Code Reusability: Write once, use many times. 2. Modularity: Break down complex problems into smaller, manageable pieces. 3. Readability: Makes code easier to understand and maintain.

□ Tip: Think of functions as mini-programs within your main program! They help keep your code clean and organized.

Common Mistakes

1. Forgetting parentheses when calling a function: `greet` instead of `greet()` (if no arguments). 2. Incorrect indentation: Python relies on indentation to define code blocks, so make sure your function body is correctly indented. 3. Not defining a function before calling it: Python executes code sequentially, so a function must be defined before it is called.

2. Parameters

Parameters are variables listed inside the parentheses in the function definition. Arguments are the values sent to the function when it is called.

```
def add_numbers(a, b): # a and b are parameters    return a + b    result = add_numbers(5, 3) # 5 and 3 are arguments    print(result)
```

Output: 8

Python supports different types of parameters:

a) Positional Arguments

Arguments passed to a function in correct positional order.

```
def describe_pet(animal_type, pet_name):    print(f'I have a {animal_type}. Its name is {pet_name}.')    describe_pet("hamster", "Harry")
```

Output: I have a hamster. Its name is Harry.

b) Keyword Arguments



Arguments identified by the parameter name. This allows you to pass arguments in any order.

```
def describe_pet(animal_type, pet_name):    print(f'I have a {animal_type}. Its name  
is {pet_name}.') describe_pet(pet_name="Willie", animal_type="dog")
```

Output: I have a dog. Its name is Willie.

c) Default Parameters

A parameter can have a default value. If the function is called without the argument, it uses the default value.

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

Output: Hello, Alice! Hello, Guest!

d) Arbitrary Arguments (*args and **kwargs)

If you do not know how many arguments will be passed into your function, add a * before the parameter name in the function definition. This way the function will receive a tuple of arguments, and can access the items accordingly.

```
def my_function(*kids):    print("The youngest child is " + kids[2])  
my_function("Emil", "Tobias", "Linus")
```

Output: The youngest child is Linus

If you do not know how many keyword arguments will be passed into your function, add two asterisks: ** before the parameter name in the function definition. This way the function will receive a dictionary of arguments, and can access the items accordingly.

```
def my_function(**kid):    print("His last name is " + kid["lname"])  
my_function(fname="Tobias", lname="Refsnes")
```

Output: His last name is Refsnes

□ Interview Question: Explain the difference between *args and **kwargs.

3. Return Statement

The `return` statement is used to exit a function and go back to the place from where it was called. It can also return a value or multiple values from the function.

```
def add(a, b):    return a + b def multiply(a, b):    result = a * b    return  
result sum_result = add(10, 5) print(f'Sum: {sum_result}') product_result =  
multiply(4, 6) print(f'Product: {product_result}')
```

Output: Sum: 15 Product: 24

If a function does not explicitly return a value, it implicitly returns `None`.

```
def do_nothing():    pass result = do_nothing() print(f'Return value: {result}')
```




Output: Return value: None

□ Tip: A function can have multiple `return` statements, but only one will be executed. The first `return` encountered exits the function.

Common Mistakes (Return Statement)

1. Forgetting to return a value when one is expected: This will result in `None` being returned, which can lead to unexpected behavior. 2. Placing code after a `return` statement: Any code after `return` in the same block will not be executed.

□ Interview Question: What is the difference between `print` and `return` in a Python function?

4. Lambda Functions

A lambda function is a small anonymous function. A lambda function can take any number of arguments, but can only have one expression.

```
x = lambda a : a + 10 print(x(5))
```

Output: 15

Why use Lambda Functions? The power of lambda is better shown when you use them as an anonymous function inside another function.

```
def myfunc(n): return lambda a : a * n mydoubler = myfunc(2) print(mydoubler(11))
```

Output: 22

□ Tip: Use lambda functions when an anonymous function is required for a short period of time.

5. Scope

A variable is only available from inside the region it is created. This is called scope.

Local Scope

A variable created inside a function belongs to the local scope of that function, and can only be used inside that function.

```
def myfunc(): x = 300 print(x) myfunc()
```

Output: 300

Global Scope

A variable created in the main body of the Python code is a global variable and belongs to the global scope. Global variables are available from within any scope, global and local.

```
x = 300 def myfunc(): print(x) myfunc() print(x)
```