Tasks – Functions Basics

Practice defining and calling functions, understanding **definition order** (Python has no hoisting: you must define a function before you call it), and using **return** to pass values back. Create each file, run it, and check the output.

Run scripts with: `python3 script_name.py`

Part 1 – Defining and calling

Task 1.1 – Define and call a simple function (`func_hello.py`)

- Create `func_hello.py` .
- Define a function named `say_hello` with no parameters: `def say_hello():` and in the body (indented) print `"Hello!"` .
- After the definition, call the function: `say_hello()` .
- Run the script.

Expected output:

```
Hello!
```

Task 1.2 – Function with one parameter (`func_greet.py`)

- Create `func_greet.py` .
- Define a function `greet(name)` that takes one parameter `name` and prints `"Hello, " + name + "!"` (or use a variable and concatenation).
- Call `greet("Alice")` and then `greet("Bob")` .

Expected output:

```
Hello, Alice!
Hello, Bob!
```

Task 1.3 – Function with two parameters (`func_add_print.py`)

- Create `func_add_print.py`.
- Define a function `add_and_print(a, b)` that takes two parameters, computes `a + b`, and **prints** the result (does not return it).
- Call it with `add_and_print(3, 5)` and then `add_and_print(10, -2)`.

Expected output:

```
8
8
```

Part 2 – Order matters (no hoisting)

In JavaScript, function declarations are "hoisted" so you can call a function before it appears in the code. In **Python**, the interpreter runs the file from top to bottom. A function must be **defined before** it is called, or you get a `NameError`.

Task 2.1 – Call before define: see the error (`func_order_wrong.py`)

- Create `func_order_wrong.py`.
- On the **first line** of the script (at the top), write a **call** to a function named `greet`: e.g. `greet("World")`.
- On the **next lines**, **define** the function: `def greet(name):` and in the body print a greeting using `name`.
- Run the script. You should get **NameError: name 'greet' is not defined** (or similar), because at the time the call runs, `greet` does not exist yet.

Expected output (you get an error):

```
NameError: name 'greet' is not defined
```

Task 2.2 – Define before call: correct order (`func_order_right.py`)

- Create `func_order_right.py`.
- **First** define the function: `def greet(name):` and in the body print a greeting using `name`.
- **Then** call it: `greet("World")`.
- Run the script. It should run without error and print the greeting.

Expected output:

```
Hello, World!
```

Part 3 – Return statements

Task 3.1 – Return a value and print it (`func_return.py`)

- Create `func_return.py`.
- Define a function `add(a, b)` that **returns** `a + b` (use the `return` keyword: `return a + b`).
- After the definition, call `add(3, 5)` and **print** the result: `print(add(3, 5))`.
- Then call and print `add(10, -1)`.

Expected output:

```
8
9
```

Task 3.2 – Return value assigned to a variable (`func_return_assign.py`)

- Create `func_return_assign.py`.
- Define a function `double(x)` that returns `x * 2`.
- Call `double(7)` and **assign** the result to a variable, e.g. `result = double(7)`.

- Print `result`. Then compute `double(10)` and assign to another variable (or reuse `result`) and print it.

Expected output:

```
14
20
```

Task 3.3 – Function with no return returns None (`func_none.py`)

- Create `func_none.py`.
- Define a function `say_hi()` that only prints `"Hi"` and has **no** `return` statement (or only `return` with nothing after it).
- Call the function and assign the result to a variable: `value = say_hi()`.
- Print `value` with `print(value)`. You should see `Hi` from the function, then `None` — in Python, a function that doesn't return anything returns `None`.

Expected output:

```
Hi
None
```

Task 3.4 – Early return (`func_early_return.py`)

- Create `func_early_return.py`.
- Define a function `describe(n)` that takes one number `n`:
- If `n < 0`, **return** immediately the string `"negative"` (e.g. `return "negative"`).
- Otherwise, return the string `"zero or positive"`.
- Call and print: `print(describe(-5))`, `print(describe(0))`, `print(describe(3))`.

Expected output:

```
negative
zero or positive
zero or positive
```

Task 3.5 – Return and use in expression (`func_return_expression.py`)

- Create `func_return_expression.py`.
- Define a function `square(x)` that returns `x * x`.
- Use the return value in an expression: e.g. `print(square(4) + square(3))` (should be 16 + 9 = 25).
- Then print `square(square(2))` (2 squared is 4, 4 squared is 16).

Expected output:

```
25
16
```

Done

You've used: `def` to define functions, parameters and arguments, **definition order** (define before call; no hoisting in Python), **return** to pass a value back, assigning the return value to a variable, functions that return `None` when they don't have a return value, and **early return** to exit a function conditionally.