

ComIT

Functions

Don't Repeat Yourself

- Don't Repeat Yourself (DRY) is an important programming principle that we try to adhere to when writing code.
 - It emphasizes reducing the amount of repeated information in a program that is likely to change.
 - The principle has been formulated by Andy Hunt and Dave Thomas in their book *The Pragmatic Programmer*.
- The basic expressions and syntax we have learned up until now does not allow us to reuse code, and this is a problem that functions solve!

What Are Functions?

- Functions are block of code that only run when we tell them too.
 - Running a function is known as “invoking it” or “executing it.”
- Functions can accept data as input (arguments) and that data can be used in the body of the function.
- Functions can also produce results that can be saved to variables. This is known as “returning” data.

Defining Functions

- In order to use functions, they must be defined before they are invoked in your code.
- Defining a function uses syntax that includes two primary parts; the **function header** that defines the “contract” or “interface” of the function, and the “**body**” of the function with the code that is executed when the function is invoked.

Defining Functions

- The header is the first line in a function definition and it starts with the keyword **def**. What follows next is a unique name provided by the programmer, a set of parenthesis, and inside the parenthesis a comma separated list of parameters.
- The Body of the function is a sequence of python statements with a consistent indentation. It can optionally end with a **return** statement.

Header —> { **def** function_name(p_1, p_2, ... p_n):
Body —> { <python statements>
 ...
 return <python expression>

Defining Functions - Parameters

- Parameters are optional variables that we define in the function header.
- They are a way for data to be passed into the body of the function.
- These are local variables during the function invocation, they do not exist before or after the function is invoked.
- Function parameters can be made optional by defining a default value in the function header:

```
def print_message(m = "Hello World!"):
    print(m)
```

Invoking Functions

- Functions that have been defined in Python can then be “invoked”.
- When a function is invoked, the code inside the function is ran.
- Functions can be invoked as many times as needed. If you find that you are repeating patterns or copy and pasting sections of code throughout your program, that is usually an indication that a function to handle that logic could be defined.

Definition { `def print_message(m = “Hello World!”):`
`print(m)`

Invocation { `print_message()`

Invoking Functions

- Functions that have been defined in Python can then be “invoked”.
- When a function is invoked, the code inside the function is ran.
- Functions can be invoked as many times as needed. If you find that you are repeating patterns or copy and pasting sections of code throughout your program, that is usually an indication that a function to handle that logic could be defined.

Definition { `def print_message(m = “Hello World!”):`
`print(m)`

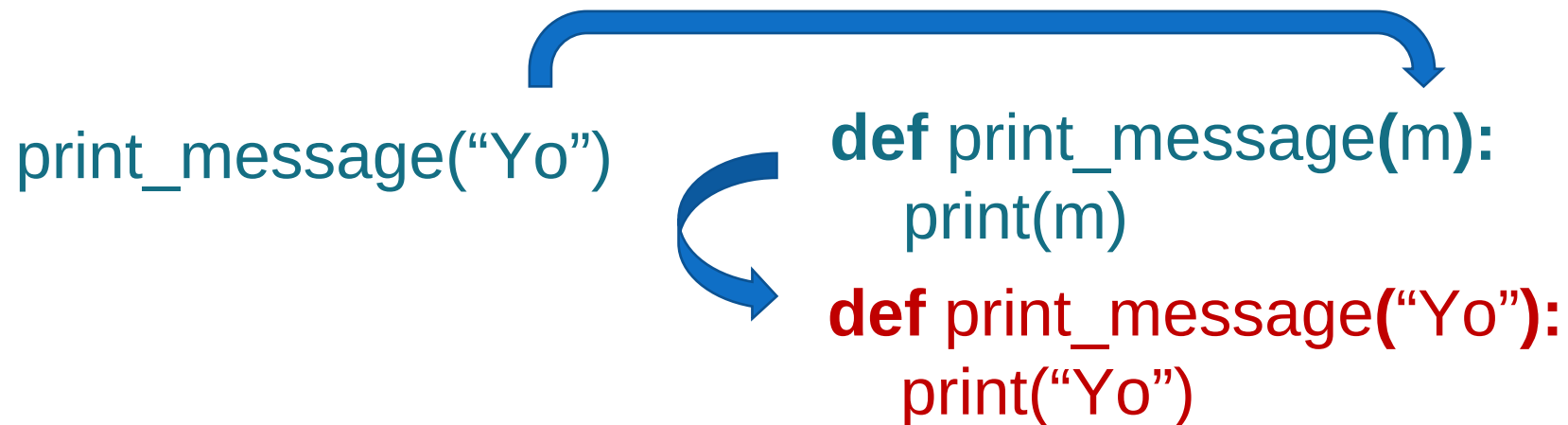
Invocation { `print_message()`

Functions – Parameter Mapping

- When supplying data to a function to satisfy parameter requirements, the data is mapped to the variables declared in the function header.
- The supplied data can be literal values, variables, or function invocations that return data of the required type.
- Data can be supplied in two ways:
 1. Positional Arguments
 2. Named Arguments


Functions – Positional Arguments

- Positional arguments are listed in the function parentheses and separated by commas.
- The arguments are mapped to the parameters in the function header based on position.
 - The first argument is supplied to the first parameter, etc.



Functions – Keyword Arguments

- Function Arguments do not need to be passed to a function based on position.
- When invoking a function and supplying arguments, the name of the parameter can be used to specify which parameter is given the argument.
- This is useful when there are many optional parameters and you only want to supply an argument to one.



```
def print_message(text="hi", output="console"):  
    ...  
    print_message(output="file")
```

Questions?