# Learn To Program With Python 3

## Control Flow - Functions

## Documentation

- Wikipedia “Control Flow” - https://en.wikipedia.org/wiki/Control_flow
- Python docs tutorial on functions: https://docs.python.org/3/tutorial/controlflow.html#defining-functions

## Control Flow

- In computer science, control flow is the order in which individual statements, instructions or function calls of a program are executed or evaluated.

# Control Flow

- Previously we have said to “think like the compiler”
    - Line by line, character by character

- “Line by Line” assumes a linear control flow, each line is executed immediately after the line above it.

- We can, and will, regularly change that.

## Control Flow

- Think of it as there being a “pointer” pointing to the current line to be executed.
- The compiler will execute the line that “pointer” is on, and then move the pointer to the next line and execute that line
  - It will continue doing this until the end of the program.
- When we first start a program, the “pointer” points at the first line and code (and goes line by line from there)
- We can tell the pointer to move to a different line of code
  - You can tell it to “jump” or “go to” different lines of your program.
  - It then resumes the normal behavior – line by line

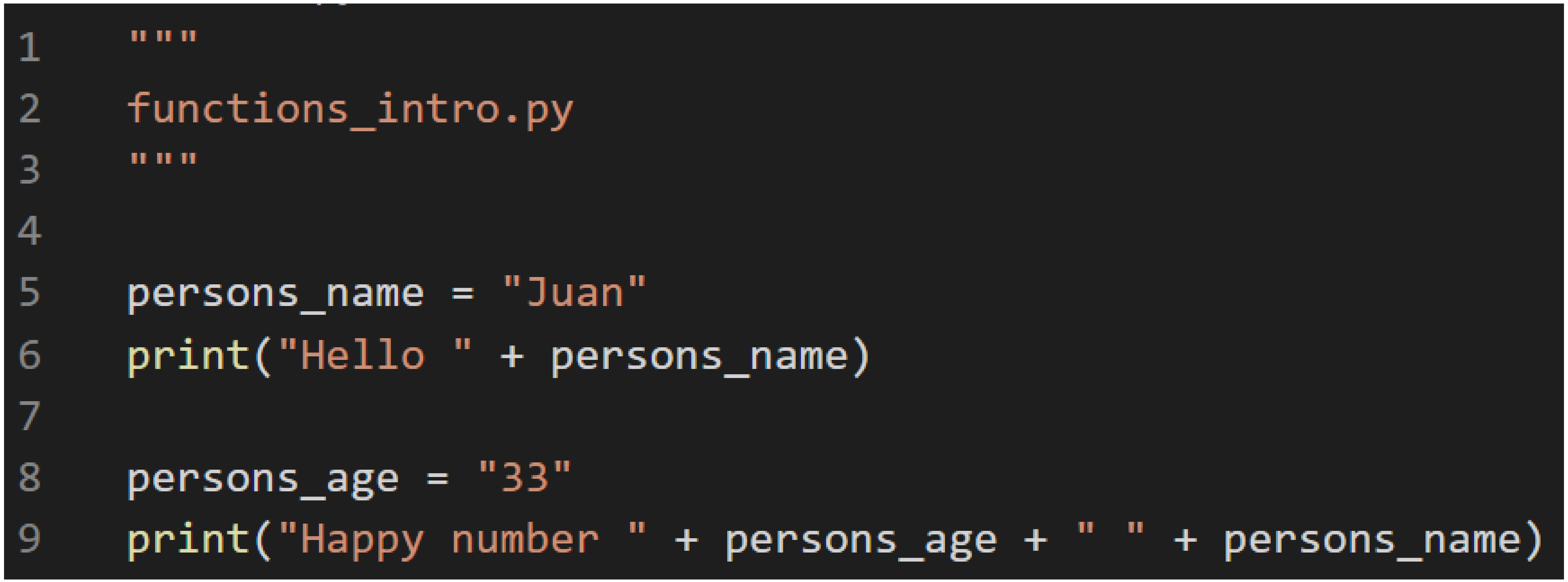
## Types of Control Flow

- Functions – reusable blocks of code referred to by a name (like a variable)
- Conditionals – execute a block of code only during certain conditions
  - If Statements (and “else” or “else if” statements)
- Loops - continuously execute a block of code until an exit condition is met
  - While loops
  - For loops
- Break / Continue statements
- Pass statements

# Linear Control Flow

- Previously we have used patterns like this.

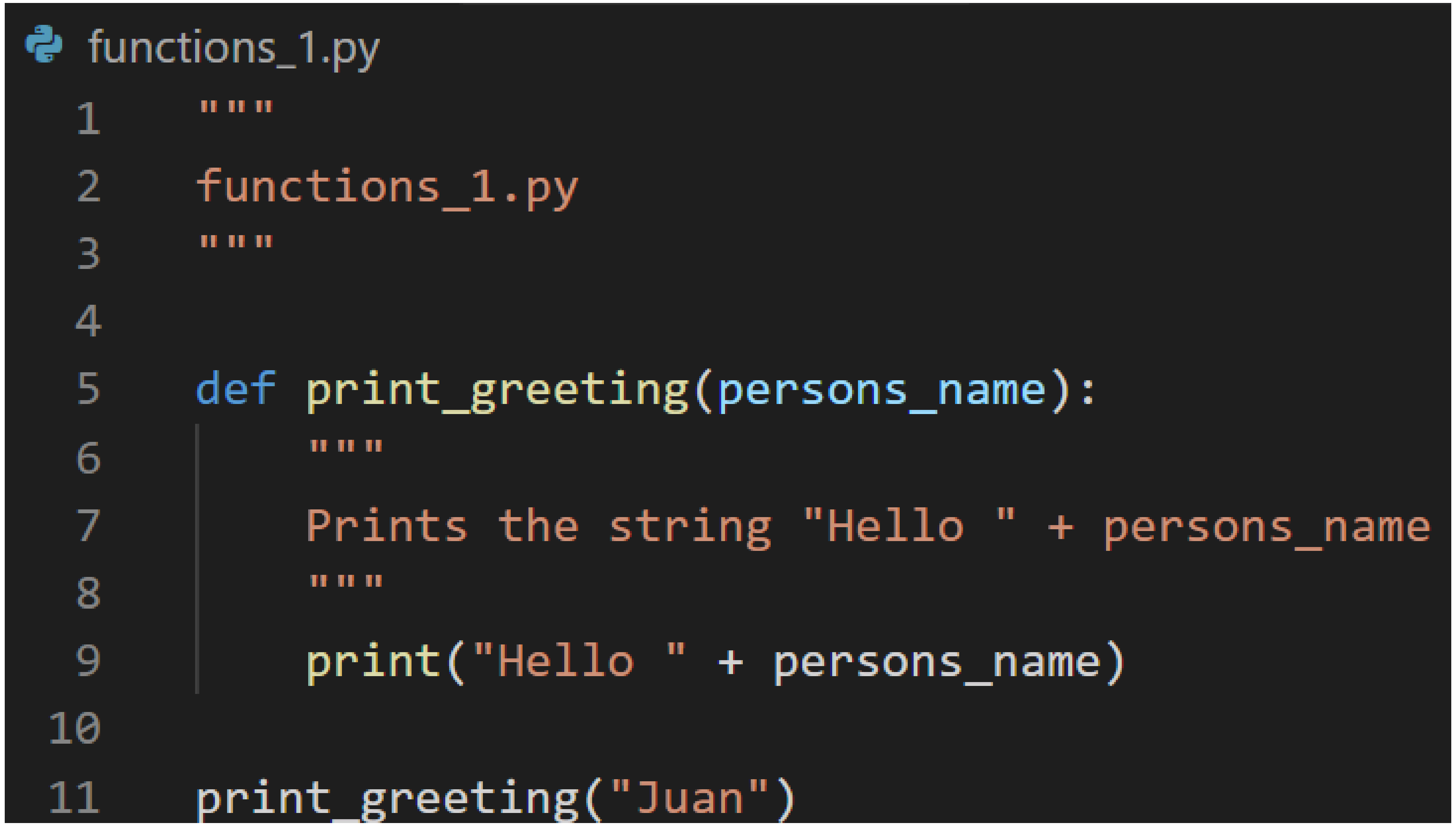
- We expect this to execute line by line, we are not altering the control flow



```python
"""
functions_intro.py
"""

persons_name = "Juan"
print("Hello " + persons_name)

persons_age = "33"
print("Happy number " + persons_age + " " + persons_name)
```

# Control Flow - Functions

- What do you think it does?
- Do you notice any patterns in the code?
- Try it!

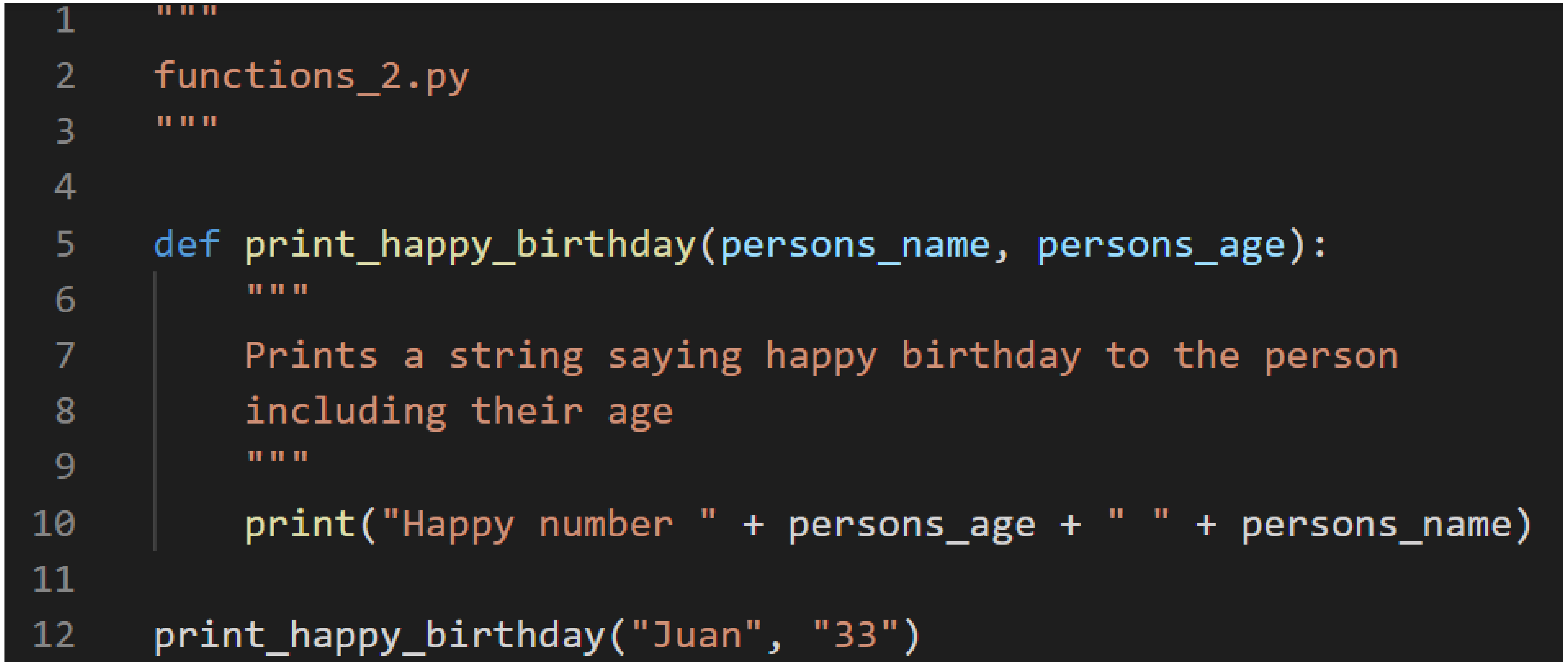


functions_1.py

```python
"""
functions_1.py
"""

def print_greeting(persons_name):
    """
    Prints the string "Hello " + persons_name
    """
    print("Hello " + persons_name)

print_greeting("Juan")
```

# Control Flow - Functions

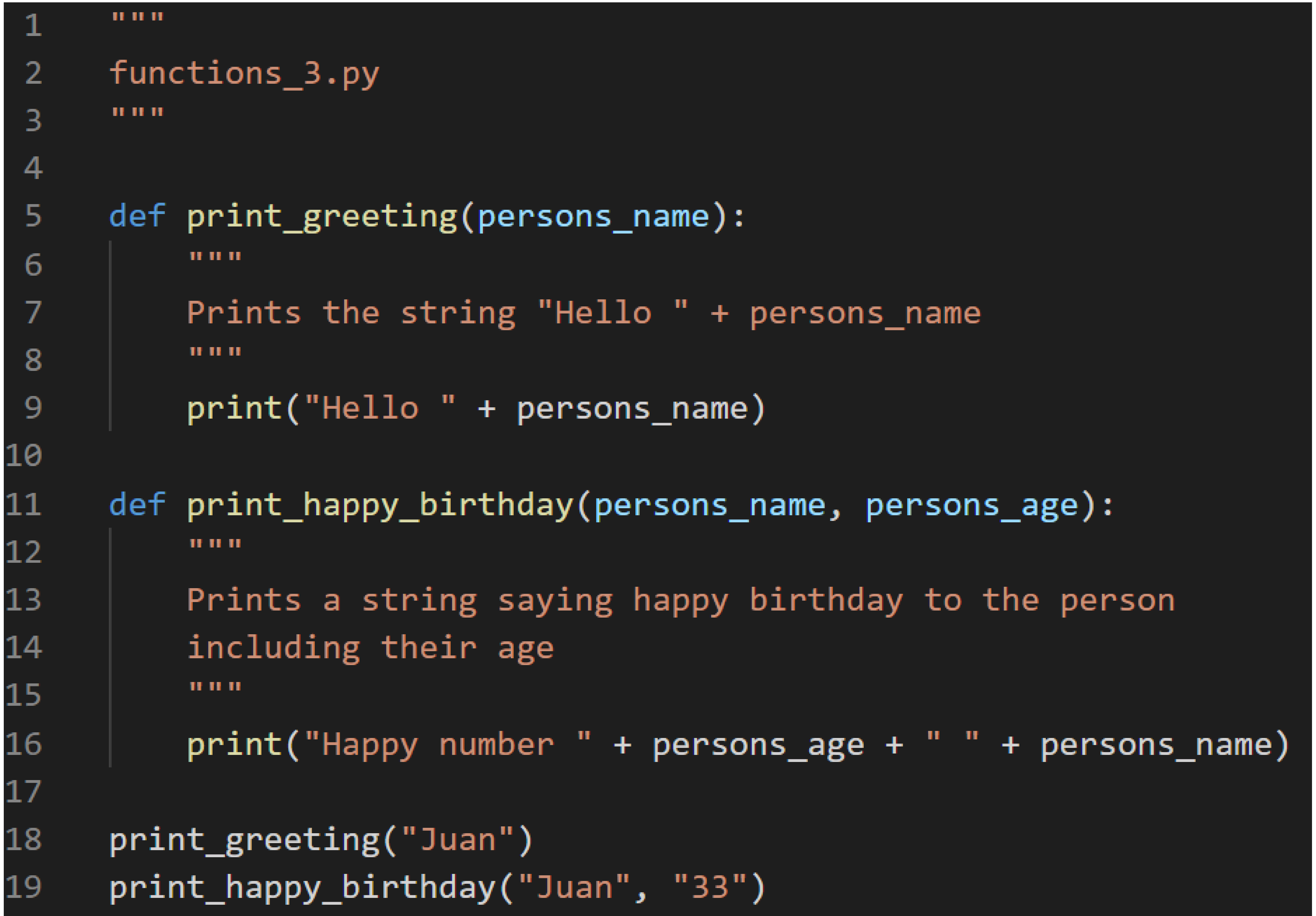
- What do you think it does?
- Do you notice any patterns in the code?
- Try it!



```python
"""
functions_2.py
"""

def print_happy_birthday(persons_name, persons_age):
    """
    Prints a string saying happy birthday to the person
    including their age
    """
    print("Happy number " + persons_age + " " + persons_name)

print_happy_birthday("Juan", "33")
```

# Control Flow - Functions

- Try it!



```python
"""
functions_3.py
"""

def print_greeting(persons_name):
    """
    Prints the string "Hello " + persons_name
    """
    print("Hello " + persons_name)

def print_happy_birthday(persons_name, persons_age):
    """
    Prints a string saying happy birthday to the person
    including their age
    """
    print("Happy number " + persons_age + " " + persons_name)

print_greeting("Juan")
print_happy_birthday("Juan", "33")
```

# Exercise!

- We will talk about the syntax of functions in a second, but first....

- Try to write a function that prints three lines of your favorite song

## Functions

In computer programming, a function is a sequence of program instructions that performs a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed.

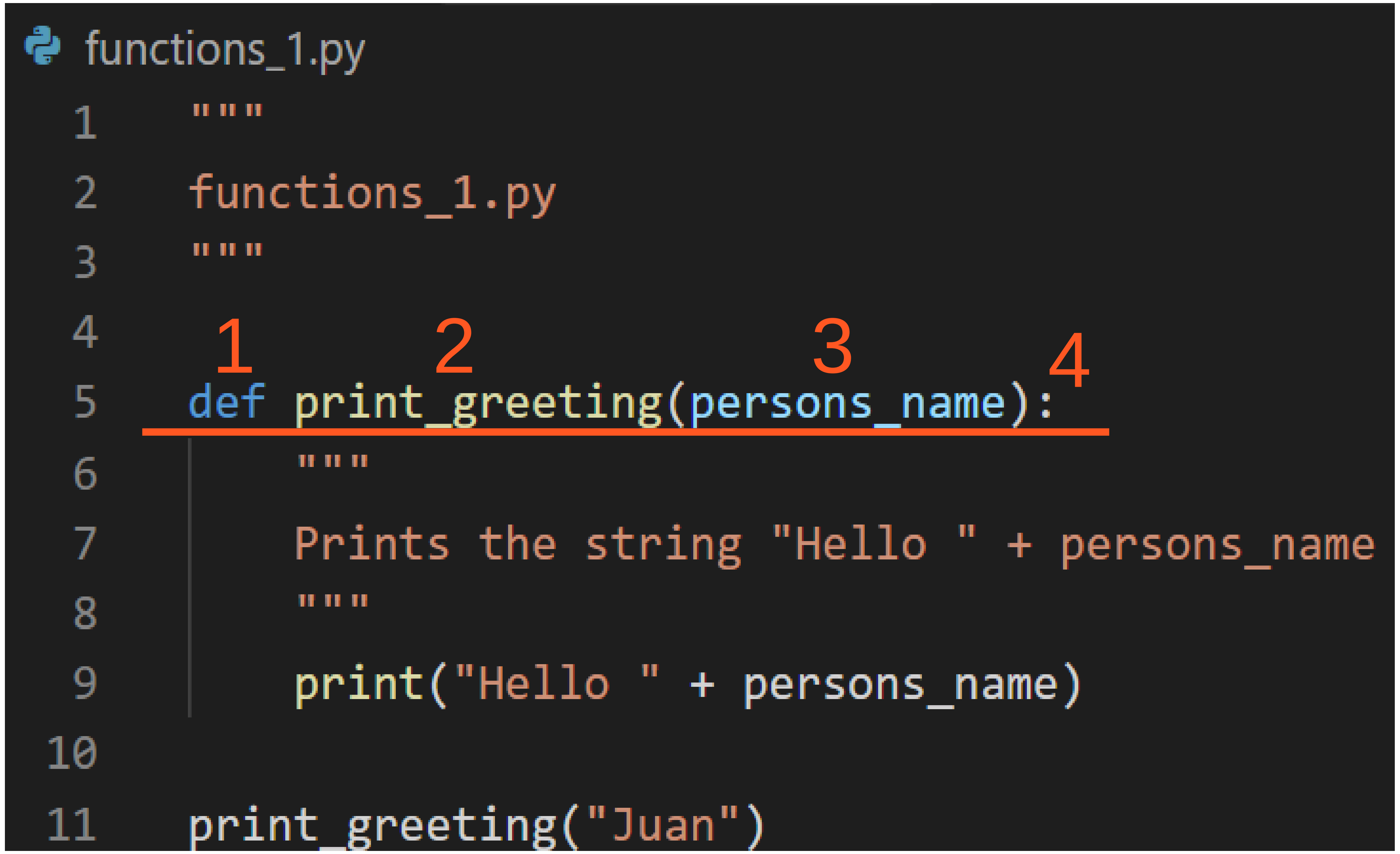
- Think of functions of “mini programs inside your program”
- Used to keep large programs organized in smaller easy to manage (and debug) sections.
- Eliminates repetitive code, makes our program smaller

## Function Syntax - Signature

1. “def” keyword defines a new function
2. “def” must be followed by the function name
   - The name is like a variable name, you can name it whatever you want and refer to the function in other parts of your code using that name
3. The function name must be followed by a parenthesized list of parameters
   - List of parameters can be empty () or a comma-separated list (parameter1, parameter2) of expressions
4. A colon : must follow the list of parameters
5. This syntax is commonly known as the “function signature”

# Function Syntax

1. “def” keyword
2. function name
3. parenthesized list of parameters
4. Colon terminates the function signature



```python
"""
functions_1.py
"""

def print_greeting(persons_name):
    """
    Prints the string "Hello " + persons_name
    """
    print("Hello " + persons_name)

print_greeting("Juan")
```

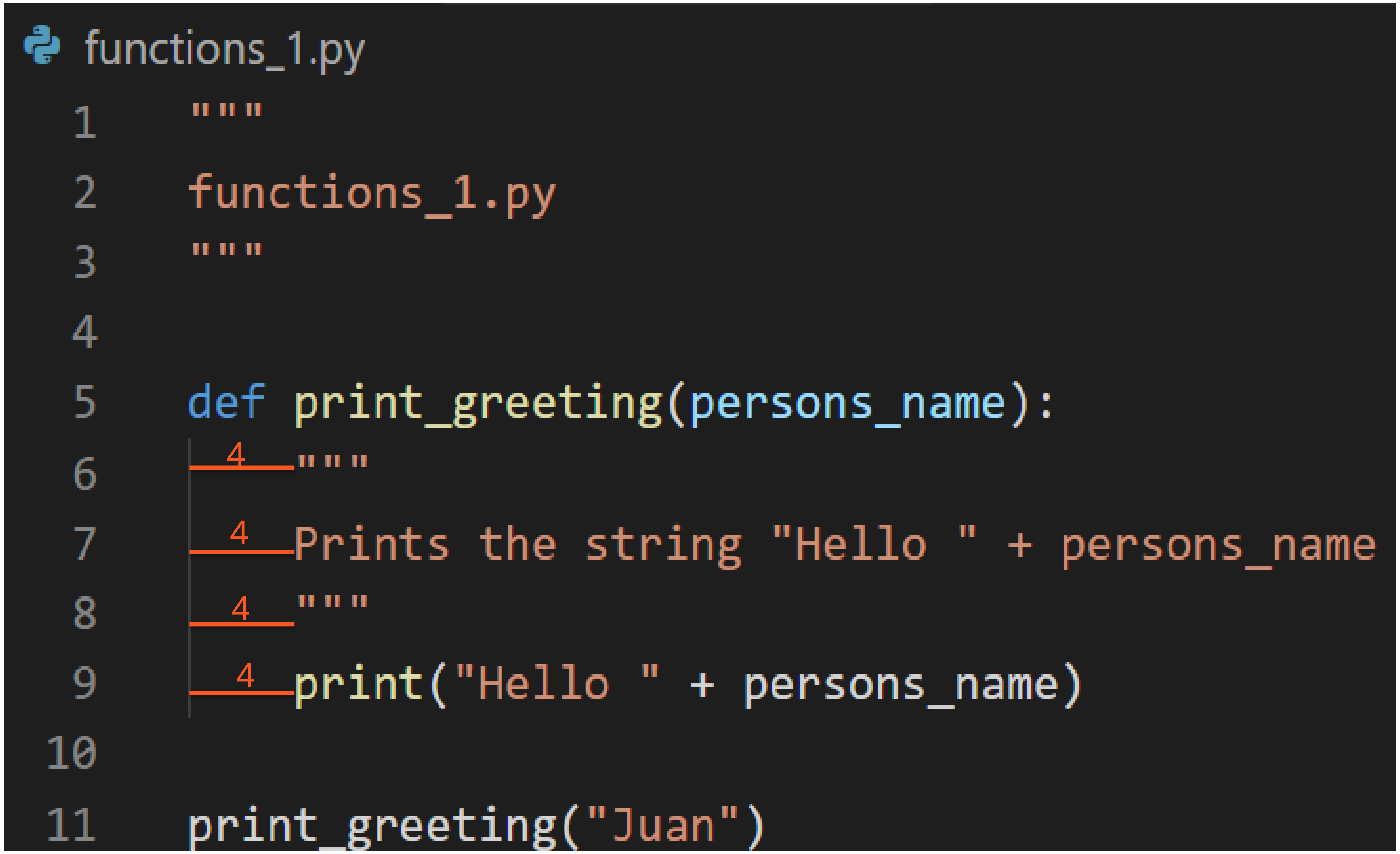
# Function Syntax - Body

1. The sequence of instructions you want the function to perform are known as the "function body"

2. The statements that form the body of the function start at the next line (after the function signature ending colon) and **must be indented**.
   - Indentation throughout your program can be done using tabs or spaces
   - But you can not mix the two!
   - Pro Tip: Never use tabs for indentation, **always use four spaces**

# Function Syntax

1. Each line in the function body is indented by four spaces


functions_1.py

```
"""
functions_1.py
"""

def print_greeting(persons_name):
    """
    Prints the string "Hello " + persons_name
    """
    print("Hello " + persons_name)

print_greeting("Juan")
```

# Function Syntax – Calling Functions

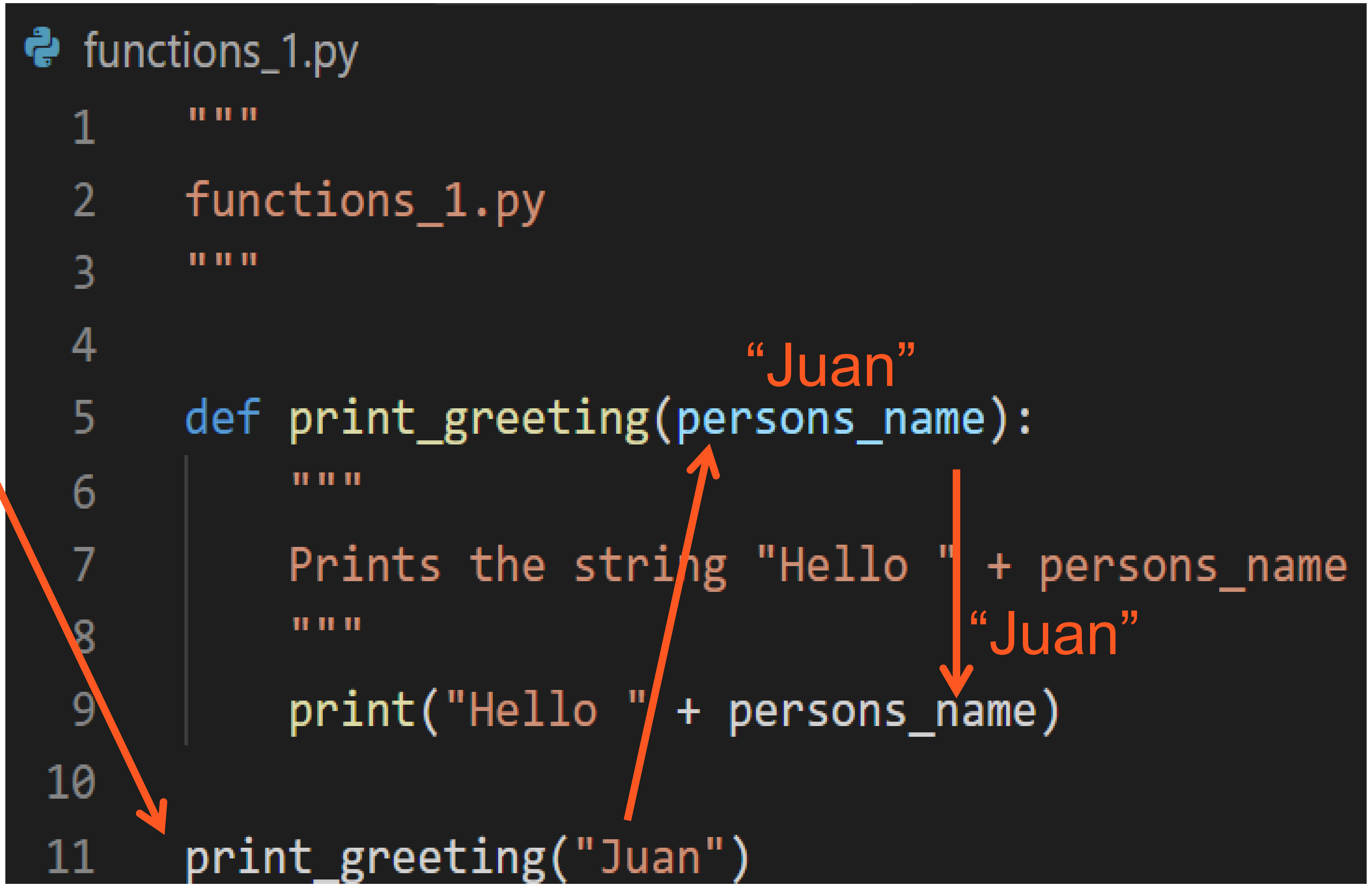
1. When you want to use a function you "call" it

2. A function call is similar to the function signature EXCEPT:
   - You do not use the "def" keyword (you are calling an already defined function, not defining a new one)
   - You do not include the colon (again, you are calling an already defined function, not defining a new one)
   - You **DO** still include the parenthesized parameter list
     - This is what actually makes the function call execute
       - Changes the control flow "pointer" to move to the function body
     - Whatever expressions you include in the parameter list will be used inside the function body

# Function Syntax

1. Function Call

2. The parameters in the function signature take on the value of the passed in parameters (arguments) from the function call



functions_1.py

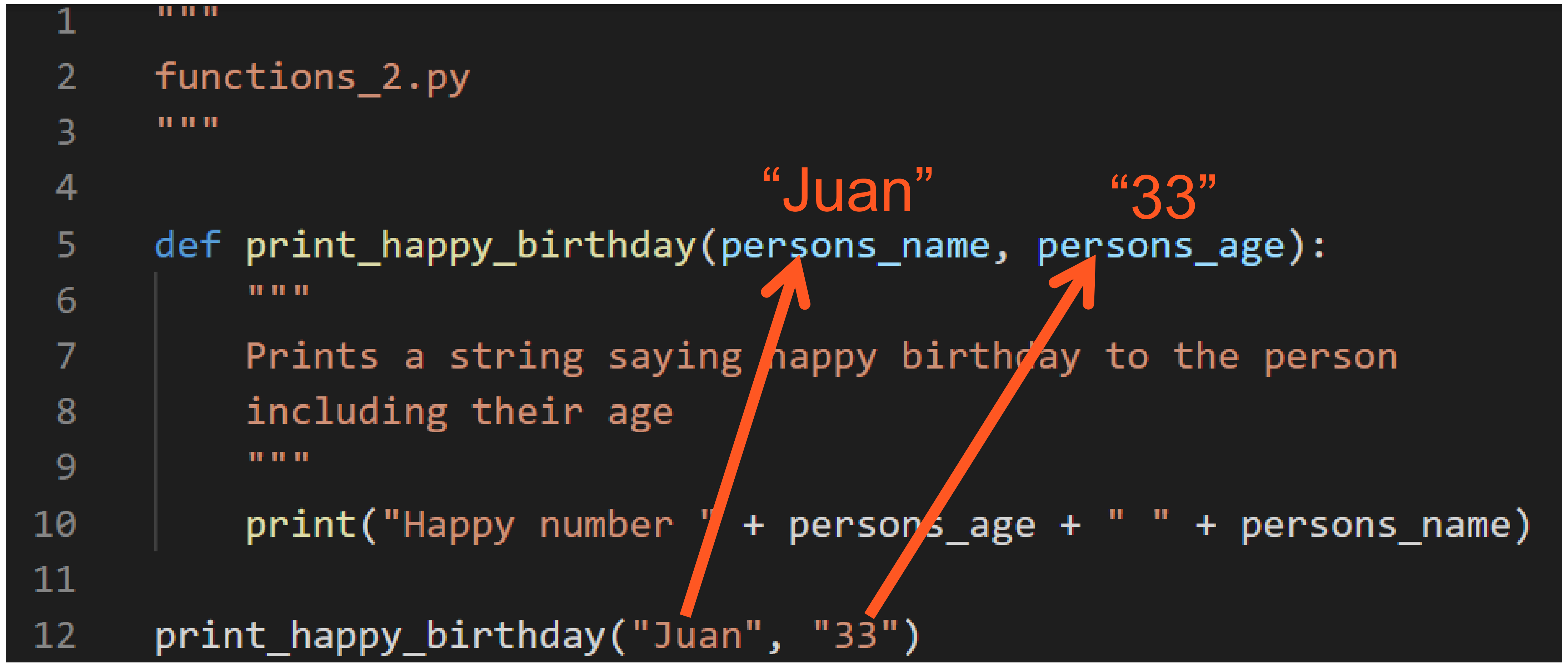
```python
"""
functions_1.py
"""

def print_greeting(persons_name):
    """
    Prints the string "Hello " + persons_name
    """
    print("Hello " + persons_name)

print_greeting("Juan")
```

# Function Syntax

1. When using a function with multiple parameters, the parameters takes on the value of the calling arguments in POSITIONAL ORDER

    1. "persons_name" is the **first** positional parameter – takes on the value of the **first** argument in the calling function – "Juan"

    2. "persons_age" is the **second** positional parameter – takes on the value of the **second** argument – "33"



```
"""
functions_2.py
"""

def print_happy_birthday(persons_name, persons_age):
    """
    Prints a string saying happy birthday to the person
    including their age
    """
    print("Happy number " + persons_age + " " + persons_name)

print_happy_birthday("Juan", "33")
```

## Exercise!

- Write a function that prints the same thing twice, it will need one parameter (the value to print)

- Write a function to square a number (multiply it by itself), it will only need one parameter (the number to square). The function should print the result.

- Write a function to exponentiate a number, it will need two parameters. The function should print the result of the first parameter raised to the power of the second parameter.

# Learn To Program With Python 3

Learn Full Stack Web Development

**www.projectfullstack.com**