DAY 10: Functions with outputs

28/01/2025: By the end of the day we're going to make a calculator and finally understand how functions really work.

Functions with Outputs

- Think of a function as a machine:
  - **Input**: Raw materials (e.g., an empty bottle).
  - **Process**: Code or transformation inside the machine.
  - **Output**: Final result (e.g., a filled bottle).

```python
def format_name(f_name, l_name):  1 usage
    formated_f_name= f_name.title()
    formated_l_name= l_name.title()

    print(f"{formated_f_name}, {formated_l_name}")

format_name( f_name: "val",  l_name: "kiyg")
```

- The `.title()` method is used to capitalize the first letter of each word.

```python
def echo_text(text):  1 usage
    return text + text

def title_case(text):  1 usage
    return text.title()

result = title_case(echo_text("hello"))
print(result)  # Outputs: Hellohello
```

- This works because `return` outputs the result, which can then be passed into another function.

Multiple return values

In this lesson, let's see what happens when a function has more than one return statement. You can have **more than one** "return" in a function. But the robot will stop at the first one it runs into and leave the function immediately.

For example:

task.py

A newer version of this course is available. We strongly recon

```python
def check_number(number):
    if number < 0:
        return "Negative number"
    elif number == 0:
        return "Zero"
    return "Positive number"
```

- If the input is `-5`, the robot stops at `return "Negative number"` and exits.
- If the input is `0`, it stops at `return "Zero"`.
- Otherwise, it goes to the last `return`.

If you just write `return` without anything after it, it's like telling the robot:
"Just stop and go home, don't bother giving me anything."

For example:

```python
def greet(name):
    if name == "":
        return  # If the name is empty, stop immediately.
    return f"Hello, {name}!"
```

If you call `greet("")` (empty name), the robot stops and doesn't do anything further.

Example with Empty Strings

Let's fix a function that formats names. If someone doesn't give their first or last name, we don't want it to keep running:

```python
def format_name(f_name, l_name):
    if f_name == "" or l_name == "":  # If either name is empty
        return "You didn't provide valid inputs."  # Exit early with a message.
    return f"{f_name.title()} {l_name.title()}"  # Format and return the full name.
```

Now, if someone gives an empty name:

```python
result = format_name( f_name: "",  l_name: "")  # Empty inputs
print(result)  # Output: "You didn't provide valid inputs."
```

But if they give valid names:

```python
result = format_name( f_name: "john",  l_name: "doe")
print(result)  # Output: "John Doe"
```

Imagine asking your robot to bake a cake. If there's **no flour**, you don't want it to keep mixing the eggs and sugar—it's a waste of time! Early return makes sure the robot stops right away if something's wrong.

Coding Exercise 10: Leap Year

exercise.py

```python
def is_leap_year(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return "True"
    return "False"


print(is_leap_year(2024))

                                # Write your code here.
                                 # Don't change the function name.
```

Docstrings

The docstring has to go as the first line after the declaration. Docstrings are **multi-line strings** used to document code in Python. They describe what a function, class, or module does, helping other developers (or your future self) understand its purpose.

Why Use Docstrings?

- They provide built-in documentation for your functions.
- When calling a function, you can view its docstring using `help(function_name)`.
- They allow for **multi-line explanations**, unlike regular comments (#).

How to Write a Docstring?

1. The **first indented line** inside a function should be the docstring.
2. Use **triple double quotes (`"""`)** to write multi-line text.

When we use a docstring, we can write as many lines as we want, and it will be interpreted all as the same thing all together, as if it was fitted onto the same line.

```python
def format_name(f_name, l_name):  1 usage
    """
            Takes a first and last name and formats them
            to return the title-cased version of the name.
            """
    formated_f_name = f_name.title()
    formated_l_name = l_name.title()
    return f"{formated_f_name} {formated_l_name}"


formatted_name = format_name( f_name: "AnGeLa",  l_name: "YU")

length = len(formatted_name)
```

Now that we've added our docstring, it's time to see what it looks like. Now, if I call this function, you can see that the text we wrote here now gets populated in the documentation.

Viewing a Docstring:

Call `help(function_name)` to see the documentation

```python
help(format_name)
```

Difference Between Docstrings and Comments:

- **Comments (#)**: Explain specific lines of code but **don't** get stored as documentation.
- **Docstrings (`"""  """`)**: Provide structured documentation and can be accessed using `help()`

The Calculator Project

```python
from art import logo
calculating = True
while calculating:
    n1 = int(input("Whats the first number ?"))
    operation = input("Whats the operation ? \n " "+ \n""- \n" "* \n" "/ \n" )
    n2 = int(input("Whats the second number ?"))


    def add(n1, n2):
        return n1 + n2
    def subtract (n1, n2):  1 usage
        return n1 - n2
    def multiply(n1, n2):  1 usage
        return n1 * n2
    def divide(n1, n2):  1 usage
        return n1 / n2
    calculator_dictionary = {
    "+": add,
    "-": subtract,
    "*": multiply,
    "/": divide,
    }
```

```python
    if operation == "+" or "-" or "*" or "/" :
        result = calculator_dictionary[operation](n1,n2)
        print(str(result))
    calculate_again = input("You want to calculate again" " yes or no?  ")
    if calculate_again == "yes" or calculate_again == "y":
        calculating = True
    else:
        calculating = False
```