Tutorial III.II - Functions

Programming: Everyday Decision-Making Algorithms

Introduction

Imagine you're a librarian who needs to organize and retrieve books efficiently. Instead of explaining every step of finding a book each time, wouldn't it be nice to have a set of standard procedures? In programming, **these procedures are called functions!** They help us organize our code into **reusable blocks**, just like having standard procedures in a library.

This tutorial will teach you about functions using a **library management system** and **a coffee machine** as an example. We'll create functions to help us manage books, track their popularity, and implement a simple caching system.

Real-World Example

Think of a function like a coffee machine:

- Input (parameters) ▯ Coffee beans, water, settings
- Process ▯ Internal brewing steps
- Output (return value) ▯ Your finished coffee

Section 1 - Creating Basic Functions

Functions in Python are like recipes:

- They have a name
- Might need ingredients (parameters)
- Follow steps (code)
- Usually produce something (return value)

Let's start with a simple function:

```python
# A simple function to greet library visitors
def greet_visitor(name):
    return f"Welcome to the library, {name}!"

# Using our function
message = greet_visitor("Tobias")
print(message)
```

```
Welcome to the library, Tobias!
```

The structure is:

1. The `def` keyword to start defining a function
2. A function name (like `greet_visitor`)
3. Parameters in parentheses (like `name`)
4. A colon `:` to start the function body
5. Indented code that makes up the function
6. Optional `return` statement to send back a value from the function

Return Value

The `return` statement is optional. If you don't use it, the function will return `None` by default.

The following illustrates this in a more generic form:

```python
def function_name(parameter1, parameter2):  # ← Function signature
    result = parameter1 + parameter2        # ← Processing
    return result                           # ← Return value
```

Common Function Errors

- `IndentationError`: Check your function's indentation
- `NameError`: Make sure the function is defined before calling it
- `TypeError`: Verify you're passing the correct number of arguments

Exercise 1.1 - Create a Book Information Function

Create a function called `get_book_info` that takes three parameters: `title`, `author`, and `year`. It should return a formatted string with the book's information.

ℹ Before You Start

Common mistakes to avoid:

- Don't forget the spaces in the formatting
- Remember to use f-strings for easy string formatting
- Make sure to include the parentheses around the year

```
# If you call the function like this:
# get_book_info("Return", "Alan Wake", 2023)
# it should return the string:
# "Return by Alan Wake (2023)"
# YOUR CODE BELOW
```

```
# Assertions to verify your solution
assert get_book_info("1984", "George Orwell", 1949) == "1984 by George Orwell (1949)"
print("Great job! Your book information function works as expected!")
```

Section 2 - Functions with Decision Making

Functions can contain any code we've learned before, including if statements. This is useful for making decisions within our functions:

```python
def is_book_available(copies_available, on_hold):
    if copies_available > 0 and not on_hold:
        return True
    else:
        return False

# Test the function
print("The book is available:", is_book_available(3, False))
print("The book is available:", is_book_available(0, False))
```

```
The book is available: True
The book is available: False
```

Here we use an `if` statement to check if the book is available. If the book is available, we return `True`, otherwise we return `False`.

Common Pitfall

When using multiple conditions, be careful with the order of operations. Consider using parentheses to make your logic clear:

```python
if (copies_available > 0) and (not on_hold):  # More readable than the code above
```

Exercise 2.1 - Create a Function to Check if a Book is Popular

Create a function called `is_popular` that takes a single parameter `rating`. If the rating is greater than 3, it should return `True`, otherwise it should return `False`.

```python
# YOUR CODE BELOW
```

```python
# Assertions to verify your solution
assert is_popular(4) == True, "A rating of 4 or higher is considered popular"
assert is_popular(2) == False, "A rating of 3 or lower is not popular"
print("Excellent! Your popular book function works correctly!")
```

Exercise 2.2 - Create a Book Access Function

Many libraries have borrowing policies that depend on multiple factors. Let's implement a realistic borrowing policy function that a library might use. Create a function called `can_borrow_book` that takes three parameters:

- `is_member`: boolean indicating if the person is a library member
- `books_borrowed`: number of books currently borrowed
- `has_overdue`: boolean indicating if they have overdue books

The function should return `True` only if:

- They are a member AND
- They have borrowed less than 3 books AND
- They have no overdue books

```
# YOUR CODE BELOW
```

```
# Assertions to verify your solution
assert can_borrow_book(True, 2, False) == True, "Should allow borrowing"
assert can_borrow_book(True, 3, False) == False, "Should not allow more than 3 books"
assert can_borrow_book(True, 1, True) == False, "Should not allow with overdue books"
print("Excellent! Your borrowing function works as it should!")
```

Section 3 - Functions with Default Parameters

Sometimes we want functions to have default values for parameters. This makes them more flexible, as we don't have to specify all parameters every time we call the function.

```python
def calculate_fine(days_overdue, rate_per_day=0.50):
    return days_overdue * rate_per_day

# Using default rate
print(f"The fine for 5 days overdue with the default rate is {calculate_fine(5)}")

# Using custom rate
print(f"The fine for 5 days overdue with a custom rate is {calculate_fine(5, 1.00)}")
```

```
The fine for 5 days overdue with the default rate is 2.5
The fine for 5 days overdue with a custom rate is 5.0
```

Exercise 3.1 - Create a Cache Priority Function

Create a function called `calculate_cache_priority` that helps decide if a book should be kept in the easily accessible section. It should take:

- `times_borrowed`: How many times the book was borrowed
- `days_since_last_borrow`: Days since last borrowed
- `is_reference`: Whether it's a reference book **(default: False)**

Return a priority score calculated as:

- For regular books: `times_borrowed / (days_since_last_borrow + 1)`
- For reference books: `2 * (times_borrowed / (days_since_last_borrow + 1))`

```python
# YOUR CODE BELOW
```

```python
# Test your answer
assert calculate_cache_priority(10, 5) == 10/(5+1), "Check your regular book calculation"
assert calculate_cache_priority(10, 5, True) == 2*(10/(5+1)), "Check your reference book
 ↪  calculation"
print("Great! Your cache priority function works and calculates correctly!")
```

Section 4 - Introduction to Methods

Methods are special types of functions that belong to objects. While regular functions **stand alone**, methods are **always associated with a specific object or data type**. Let's explore this with some examples:

```
# String methods
book_title = "the great gatsby"
print(book_title.title())  # Outputs: "The Great Gatsby"
print(book_title.upper())  # Outputs: "THE GREAT GATSBY"
```

```
The Great Gatsby
THE GREAT GATSBY
```

```
# List methods
books = ["Neuromancer", "1984", "Foundation"]
books.append("Dune")  # Adds a book to the list
print("Books before sorting:", books)
books.sort()          # Sorts the list alphabetically
print("Books after sorting:", books)
```

```
Books before sorting: ['Neuromancer', '1984', 'Foundation', 'Dune']
Books after sorting: ['1984', 'Dune', 'Foundation', 'Neuromancer']
```

Functions vs Methods

The main difference:

- Functions are called directly: `function_name(arguments)`
- Methods are called on objects: `object.method_name(arguments)`

Exercise 4.1 - Working with String Methods and Functions

Create a function called `format_book_title` that takes a book title as input and:

1. Capitalizes the first letter of each word
2. Removes any extra whitespace
3. Returns the formatted title

Common String Methods

- `.strip()` - Removes whitespace from start and end
- `.title()` - Capitalizes first letter of each word
- `.lower()` - Converts to lowercase
- `.upper()` - Converts to uppercase

```
# YOUR CODE BELOW
```

```
# Test your answer
assert format_book_title("  the hitchhikers guide  ") == "The Hitchhikers Guide", "Check
 ↪  your whitespace removal"
assert format_book_title("dune") == "Dune", "Check your capitalization"
print("Excellent! Your title formatting function works correctly!")
```

Conclusion

Excellent work! You've learned how to create and use functions in Python through the lens of library management. You now know how to:

- Create basic functions with parameters
- Use functions with decision-making logic
- Work with default parameters
- Return values from functions

Remember:

- Functions help us organize and reuse code
- They make our code more readable and maintainable
- They can take inputs (parameters) and produce outputs (return values)
- They can have default values for parameters

In a real library system, functions like these would be part of a larger caching system that helps optimize book retrieval times and improve user experience!

Solutions

You will likely find solutions to most exercises online. However, we strongly encourage you to work on these exercises independently without searching explicitly for the exact answers to the exercises. Understanding someone else's solution is very different from developing your own. Use the lecture notes and try to solve the exercises on your own. This approach will significantly enhance your learning and problem-solving skills.

Remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities. If you encounter difficulties, review the lecture materials, experiment with different approaches, and don't hesitate to ask for clarification during class discussions.

Later, you will find the solutions to these exercises online in the associated GitHub repository, but we will also quickly go over them next week. To access the solutions, click on the Github button on the lower right and search for the folder with today's lecture and tutorial. Alternatively, you can ask ChatGPT or Claude to explain them to you. But please remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities.

In the next tutorial, we'll take a look on how to import packages and libraries to extend our functionality!