CSE 160: Functions and Abstraction

Instructor: Alessia S. Fitz Gibbon

Autumn 2024

Introduction to Functions

In mathematics, functions allow us to map an input to an output. In Python, functions are a way to group statements together so they can be executed more than once. You have already used Python's built-in functions like `len`, `range`, and `math.sqrt`. Functions promote code reuse, reduce redundancy, and increase readability.

In Python, you can define your own functions, allowing you to package logic for repeated use. Functions take parameters as input, perform a computation, and return an output.

Example:

```
def square(x):
    return x * x
```

In this function, `square` takes one argument `x` and returns the square of the input.

Function Calls

Calling a function means executing the code inside it. In Python, you can call a function like this:

```
result = square(5)
print(result) # Output: 25
```

Functions can be called with various types of arguments, including integers, floats, and even other function calls.

Practice Problem 1:

Write a Python function `cube(x)` that returns the cube of its argument `x`. Call this function for different values and print the results.

Functions as Abstractions

Functions are not just useful for code reuse; they help abstract away complexity. You don't need to know how the function works, just what it does. This is known as functional abstraction.

```
def fahr_to_celsius(fahr):
    return (fahr - 32) * 5 / 9
```

The above function converts Fahrenheit to Celsius, but when you call it, you don't need to worry about the specific math behind the calculation.

Practice Problem 2:

Define a function `celsius_to_fahr(celsius)` that converts Celsius to Fahrenheit using the formula $\texttt{fahr} = \frac{9}{5} \times \texttt{celsius} + 32$. Use this function to convert 100 degrees Celsius to Fahrenheit.

Parameter Passing

When you call a function, the values you pass are known as arguments. These arguments are assigned to the function's formal parameters. The parameters act as variables inside the function. Python supports passing arguments by value. Example:

```
def greet(name):
    print("Hello, " + name + "!")
```

Here, the argument `"Alice"` is passed to the function:

```
greet("Alice")  # Output: Hello, Alice!
```

Practice Problem 3:

Write a function `calculate_area(length, width)` that calculates and returns the area of a rectangle. Call the function for a rectangle with length 5 and width 10, and print the result.

Return Values

Functions in Python can return values to the caller using the `return` statement. A function can also perform operations without returning a value, but in most cases, you will want to return a result.

Example:

```
def add(x, y):
    return x + y
```

The function returns the sum of `x` and `y`.

Practice Problem 4:

Define a function `max_of_three(a, b, c)` that returns the maximum of three numbers. Use this function to find the maximum of 5, 10, and 8.

Practice Problem 5:

Study the following simple code with one global and one local variable. Answer the questions below.

```
x = 10  # Global variable

def my_function():
    x = 3  # Local variable
    print("Inside the function, x =", x)

my_function()
print("Outside the function, x =", x)
```

1. What is the value of `x` inside the function?

2. What is the value of `x` outside the function?

Conclusion

Functions play an essential role in writing clean, maintainable, and reusable code. Understanding how to define and call functions is foundational to becoming proficient in Python programming. Practice defining functions that perform various tasks, passing different kinds of arguments, and working with return values.