2. Functions

*Notes generated by Claude 3 Sonnet to more comprehensiveness

What is a Function?

A function is a reusable block of code that performs a specific task. It encapsulates a set of instructions and can be called (or invoked) from different parts of your program whenever you need to execute those instructions. **Functions help to modularisation code, making it more organised, easier to maintain, and promote code reuse.**

Built-In Functions

Python comes with a vast collection of built-in functions that are readily available for use. These functions are pre-written and designed to perform specific tasks, saving you from writing redundant code. Here are a few examples of built-in functions and their usage:

1. **print()**: This function is used to display output to the console or terminal.

```
print("Hello, World!")  # Output: Hello, World!
```

2. **len()**: This function returns the length (the number of items) of an object.

```
word = "Python"
length = len(word)
print(length)  # Output: 6
```

3. **int()**: This function converts a value to an integer data type.

```
num_str = "42"
num_int = int(num_str)
print(num_int)  # Output: 42
```

4. **str()**: This function converts a value to a string data type.

```
num = 42
num_str = str(num)
```

```
print(num_str)  # Output: "42"
```

5. **sum()**: This function calculates the sum of all items in an iterable (like a list or a tuple).

```
numbers = [1, 2, 3, 4, 5]
total = sum(numbers)
print(total)  # Output: 15
```

These are just a few examples of built-in functions in Python. There are many more functions available, and you can explore them using the Python documentation or by searching online.

Defining a Function

In Python, you define a function using the `def` keyword, followed by the function name, parentheses `()` for parameters (if any), and a colon `:`. The function body is indented by *either a tab or 4 spaces*, and contains the statements that the function will execute when called.

```
def function_name(parameters):
    """
    Docstring: A brief description of what the function does.
    """
    # Function body
    # Statements to be executed
    return value  # Optional: Return a value if needed
```

Let's break down the components:

- `def`: The keyword used to define a function.
- `function_name`: The name you give to the function, following Python's naming conventions (lowercase with words separated by underscores).
- `parameters` (optional): A comma-separated list of input values that the function can accept. These are placeholders for the actual values that will be passed when the function is called.
- `"""Docstring"""`: A multi-line string (enclosed in triple quotes) that provides a brief description of what the function does. This is optional but highly recommended for documenting your code.
- Function body: The indented block of code that contains the statements that the function will execute when called.
- `return` (optional): The keyword used to return a value from the function. If no value is specified after `return`, the function will return `None` by default.

Calling a Function

Once you've defined a function, you can call (or invoke) it from other parts of your program by using the function name followed by parentheses `()`. If the function requires arguments (input values), you pass them inside the parentheses, separated by commas.

```
function_name(argument1, argument2, ...)
```

Here's an example of a function that takes two numbers as input and returns their sum:

```
def add_numbers(x, y):
    """
    Add two numbers and return the result.
    """
    result = x + y
    return result

# Calling the function
sum1 = add_numbers(2, 3)  # sum1 will be 5
sum2 = add_numbers(10, 20)  # sum2 will be 30
print(sum1)  # Output: 5
print(sum2)  # Output: 30
```

Function Parameters

Functions can accept different types of parameters, including positional arguments, keyword arguments, default arguments, and arbitrary arguments.

Positional Arguments

Positional arguments are the simplest form of arguments, where the values are matched to the parameters based on their position in the function call.

```
def greet(name, age):
    print(f"Hello, {name}! You are {age} years old.")

greet("Alice", 25)  # Output: Hello, Alice! You are 25 years old.
```

Keyword Arguments

Keyword arguments allow you to specify the parameter names when calling the function, making the code more readable and less error-prone.

```
def greet(name, age):
    print(f"Hello, {name}! You are {age} years old.")

greet(age=25, name="Alice")  # Output: Hello, Alice! You are 25 years
old.
```

Default Arguments

You can provide default values for function parameters, which will be used if no argument is passed for that parameter.

```
def greet(name, age=25):
    print(f"Hello, {name}! You are {age} years old.")

greet("Alice")  # Output: Hello, Alice! You are 25 years old.
greet("Bob", 30)  # Output: Hello, Bob! You are 30 years old.
```

Arbitrary Arguments

Python also allows you to define functions that can accept an arbitrary number of arguments using the `*args` and `**kwargs` syntax.

- `*args` (non-keyword arguments): Allows you to pass a variable number of positional arguments to a function. Inside the function, `args` is treated as a tuple containing all the positional arguments.
- `**kwargs` (keyword arguments): Allows you to pass a variable number of keyword arguments to a function. Inside the function, `kwargs` is treated as a dictionary containing the keyword arguments.

```
def print_args(*args, **kwargs):
    print("Positional arguments:", args)
    print("Keyword arguments:", kwargs)

print_args(1, 2, 3, name="Alice", age=25)
# Output:
# Positional arguments: (1, 2, 3)
# Keyword arguments: {'name': 'Alice', 'age': 25}
```

Function Scope

In Python, variables have different scopes depending on where they are defined. The scope determines the visibility and lifetime of a variable.

- **Global scope**: Variables defined outside of any function or class are considered global variables. They can be accessed and modified from anywhere in the program.
- **Local scope**: Variables defined inside a function are considered local variables. They are only accessible within the function where they are defined.

```
# Global variable
x = 10

def my_function():
    # Local variable
    y = 20
    print(x)  # Access the global variable x
    print(y)  # Access the local variable y

my_function()  # Output: 10, 20
print(x)  # Output: 10
print(y)  # Error: NameError: name 'y' is not defined
```

To modify a global variable inside a function, you need to use the `global` keyword:

```
x = 10

def my_function():
    global x  # Declare that we want to modify the global variable x
    x = 20    # Modifying the global variable x

my_function()
print(x)  # Output: 20
```

See Also

3. Methods