# Session 1 - Functions

In this session we will learn all about functions in Python. In fact, we already used functions last week including `print` and `input`. Functions make coding easier, faster, and more readable for the user. We will focus on the main components in defining a function and finally how to call local functions as well as functions from Python libraries.

## 🎯 Goals

Our **goals** for today's session are to:

- identify the main components to define and call a function in Python.
- import Python libraries and call functions associated with an outside library.

## 🧑‍🏫 Lesson: Functions

We see functions used everyday in real life. Think about the activities in your life that seem somewhat automated or routine. Most likely those are powered by functions! For example, when you go to pay using a self checkout machine or maybe as simple as setting the table.

### 🔎 Key Vocabulary

- **Function**: A group of actions or lines of code that completes a task and sometimes returns an item or a collection of items. Functions are often referred to as procedures.

    - **Function signature**: Where the function is defined with its name and inputs.
    - **Function body**: Where the steps/procedure of the function is implemented and results are outputted.
- **Define**: When programmers refer to "defining" something (i.e defining a function or defining a variable) this refers to the action of **creating** the new object with the appropriate syntax rules.
- **Call**: When programmers refer to "calling" something (i.e calling a function or calling a variable) they really mean to **access or use** the object.
- **Parameters**: Inputs to a function. Some functions may have no parameters while others may have one or more parameters separated by commas.
- **Library**: A collection of functions that can be referenced or accessed.

## Functions in Python

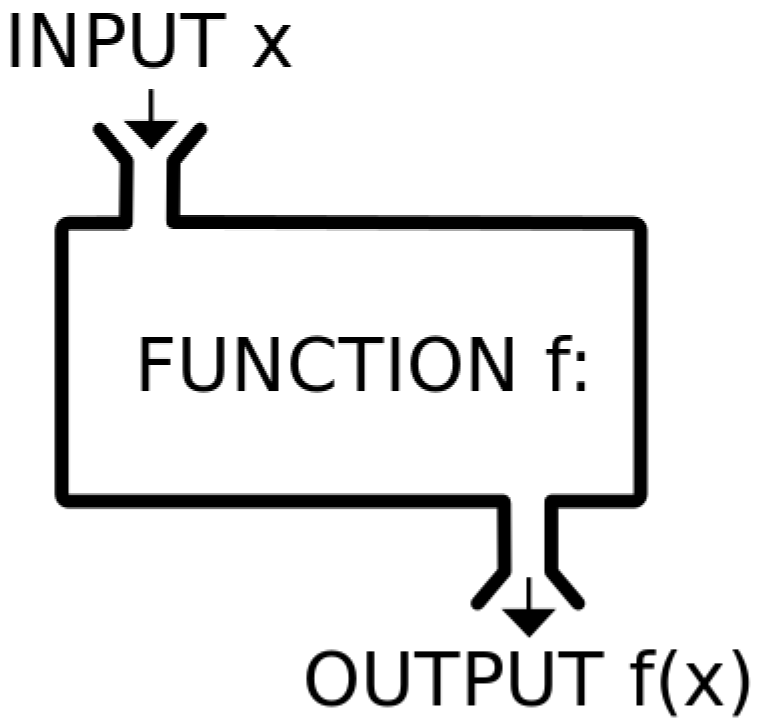
A **function** is a group of lines of code that completes a task. Programmers use functions all the time because they make it easier and faster to code and also makes code more readable to other users. Programmers use **functions** to define procedures in the code (i.e measure distance, print to the console, find a value, etc.). In a previous lesson we already saw the use of multiple built-in functions in python.

- `print`: This function takes in an object and prints it to the console.
- `input`: This function takes in a string input and prints the question to the console and returns the user's response.
- `int`: This function converts an object into an integer data type.

Python has many built-in functions that makes it easier for programmers to code. Feel free to explore the full list of Python built-in functions. Python has these built-in functions readily available to programmers so that we don't have to write them from scratch every time! This makes coding easier.

### Input and Output

A function's purpose is to execute a set of commands (or lines of code) to complete some procedure. In many cases, we can think of a function that takes in some **input** or information and does something and finally sends a response or **output**.



*Source: Wikipedia*

Let's take a look at a few examples of Python built-in functions

| **Function Name** | **Input** | **Function** | **Output** |
|---|---|---|---|
| `print` | Some object | Takes the object and converts it to a string and prints it out on the console | None. The computer only prints to the console but does not return anything back to the program. |

| | | | |
|---|---|---|---|
| `input` | A `str` data type | Takes the input and prints to the console and waits for a user to respond. | A `str` data type containing the user's response. |
| `int` | A number or `str` data type | Takes the input and converts it to an `int` data type. | A `int` data type that is equivalent to the input number |

Notice that the lines of code to execute these functions are hidden to us as users of Python. We don't need to be concerned with the details on *how* these functions are executed but we must follow the **rules** for the function to work properly. It is important that we always provide the appropriate input in the correct format. Otherwise we would get an error when we try to run the program.

‼ **Not all functions require input and/or output**. Take the `print` function as an example. It takes in a `str` input but nothing is outputted, instead we just see our results in the console. Similarly, not all functions require any input to execute their procedure. You can find functions that access stored information that may not need any input.

**Input** - We often refer to the input of a function as **parameters**. Some functions may contain no parameters or multiple parameters. There are no set limitations on how much or how little parameters a function may need. Programmers also have the option to indicate *required* and *optional* parameters when a function is called.

**Output** - We often refer to the output of a function as the **return value**. Similar to parameters, functions may return no data or multiple pieces of data. We call the output the *return* value because it often appears after the `return` keyword. We will see more examples of this when reviewing the syntax of defining a function.

## Local Functions

In some instances you will need to create **local functions** that only exist in your own specific program. Before we look into the syntax, let's review 2 terms we often hear when talking about functions: **define** and **call**. We often refer to the creation of a new function as **defining a function**. When we want to use the function, we call this **calling a function**. These terms aren't just reserved for functions. We can say that we *define a variable* which just means we create a variable.

## Defining a Function

Let's take a look at the syntax on how to create functions in your program.

**Defining a Function Syntax**

```
def func_name(param1, param2): # function signature
  doSomething # function body
  return result
```

**Function Signature**: We refer to the defining of a function or the very first line that contains the function name, input, and output of a function as the **signature**. Let's breakdown the syntax of the function signature

- `def`: This keyword is used to indicate the defining of a new function.
- `func_name`: The name of your function. In general, function names should not contain any spaces and start with lowercase letters. You can use either snake case or camel case to distinguish different words in a function name.
- `()`: Parentheses are used to indicate a function's input. Any objects inside the parentheses are called **parameters**, or inputs.
- `param1`,`param2`: `param1` and `param2` are indicators for parameter or inputs to the function. If a function has multiple parameters, they must be separated by a comma. In some cases, a function may not have any parameters. In this case, there are no objects that appear in between the parentheses `()`.
- `:`: We use a colon to indicate the **end** of the method signature. All lines of code that appear after the colon `:` is part of the body of the function.

**Function Body**: We refer to the code that is executed when the function is called as the **body**.

- `doSomething`: This is a placeholder for all of the lines of code to be executed when the function is called. All code that exists in the body must appear after the **function signature** (or after the colon `:`) and be **indented**. Indentation in Python is important to indicate program flow and blocks of code. Once the computer reaches a line of code that is not indented or reaches a `return` statement, it will conclude executing the function.
- `return`: This keyword lets the computer know when to end the function and also what is outputted. **Not all functions will have a return statement**.

## Calling a Function

Let's take a look at the syntax on how to call or access functions in your program.

**Calling a Function Syntax**

```
func_name(param1, param2)
```

- `func_name`: The function name that we want to call
- `()`: Parentheses are used to indicate we want to call, or let the computer know to run the set of code associated with a function.
- `param1`, `param2`: `param1` and `param2` are placeholders for the parameters or inputs that the function accepts. If a function has multiple parameters, they must be separated by a comma. In some cases, a function may not have any parameters. If a function has no parameters, then there will be nothing in between the parentheses.

### Defining & Calling Function Example

Let's take a look at an example of a function named `university_standing` that takes in a color name and returns the Hogwarts house name from the movie Harry Potter.

```
# Defining the university_standing function
```

```python
def university_standing (grad_year):
  if grad_year == 2022:
    return "senior"
  elif grad_year == 2023:
    return "junior"
  elif grad_year == 2024:
    return "sophomore"
  elif grad_year <= 2025:
    return "freshman"
  else:
    return "alumni"

# Defining variable year and setting to value 2023 to test
year = 2023
# Calling function university_standing with input year
university_standing(year)
```

Notice how we call our `hogwarts` function *outside* of the function body.

### 🎬 Now it's your turn!

Try writing a function of your own `guess_number` that takes in a guess from the user and returns whether or not they are higher, lower, or on target to the secret number.

- **Input:** `int` data type that represents `guess`.
- **Output:** `str` data type that is either `guess is higher`, `guess is lower`, and `guess is correct`.

## Python Libraries

Python language is always growing and changing. Libraries are written collections of functions that can be used by anyone! Why reinvent the wheel?
In fact, there exists over 137,000 libraries in Python! There are libraries to simulate game development, visualize data, process and run machine learning algorithms, make complex mathematical calculations, and so much more.
One we will explore in particular is the random library. The `random` library is a collection of functions that can generate pseudo-random numbers.
In particular we will use the `randint(a,b)` function that returns a random integer between a <= N <= b.
Let's take a look at the syntax for **importing** a library to your program or repl.

**Importing Libraries Syntax**

```
# Import library to program
import library_name
```

We typically load all of the libraries we want to use at the **top of the program**.

- `import`: This keyword is used to load the details of a library to your program
- `library_name`: The name of the library you want to use.

Next, let's look at how you can call a function from the library within your code.

**Calling Functions from a Libraries Syntax**

```
# Using functions in the library
library_name.func_name()
```

We typically load all of the libraries we want to use at the **top of the program**.

- `library_name`: First we need to specify which library we want to use/call
- `.` : We use a period to indicate that we are calling a function from the specified library.
- `func_name`: The function we want to use
- `()` : Finally we use parentheses to indicate the calling of a function. If the function has parameters (or inputs) we would also include them inside the parentheses.

## 🎬 Now it's your turn!

Update the guess_number function that randomly chooses a secret number each time that the function is called.

1. Link/Import the `random` Python library
2. Change the `secret_number` variable to the output of a `randint()` function. Let's choose a random number from **1 to 100 inclusively**.
3. Print the `secret_number` at start so we know if our program is working correctly.

💡 **Tip:** `random.randint(a, b)` returns a random integer between a <= N <= b.

## 🎬 Activity: Password Generator

It is important to use a **secure** password for any online account. Your goal is to create a simple password generator that creates an 8-character long password with a mix of letters and digits.

**Example Password Generator**

### Directions

1. Begin planning your approach for generating an 8-digit password
2. Create a new repl on Replit.
3. Start coding your password generator function.
4. Test your program by running calls to your function and analyzing the output. Be sure to test at least 3 cases

💡 **Tip**:

- Before beginning, take a moment to *plan* your function. What is the input, output, and general steps of your program.
- To generate random letters you will need to use the `string` AND `random` Python library. Try googling on your own first! There are many resources out there!

🐞 **Debugging Tips**

- When running your code, did you get a `SyntaxError`?
    - Remember that syntax refers to the set of rules we must follow in a programming language. When you receive a `SyntaxError` double check the spelling and case of any variables or functions. Don't forget that coding is very sensitive to spelling and capitalization of characters.
- Did you include parenthesis `()` when calling the `print` or `input` function?
- Did you include a pair of quotation marks `" "` or `' '` for all **String** data types?
- Did you import both the `string` AND `random` Python library at the top of your program?
- Did you indent all code blocks within your function body?

- When printing multiple values, do you use commas to separate the objects in the `print()` function?
- When calling functions from a library, did you specify the library name and use a period before calling the specific function name?
- Not sure how to generate random letters? Check out this Stack Overflow post.