Set 0: The Print Function

Skill 0.1: Explain the purpose of a built-in function
Skill 0.2: Interpret a built-in Python function
Skill 0.3: Explain the process of function invocation
Skill 0.4: Explore the print function
Skill 0.5: Apply the escape and new line characters
Skill 0.6: Use multiple arguments in a print function
Skill 0.7: Use keyword arguments in the print function

Skill 0.1: Explain the purpose of a built-in function

Skill 0.1 Concepts

Look at the line of code below,

```
print("Hello, World!")
```

The word print in the line of code above is the function name. You've probably encountered the term function many times before, during math classes. You can probably also list several names of mathematical functions, like sine or log.

Python functions, however, are more flexible, and can contain more content than their mathematical siblings.

A function (in this context) is a separate part of the computer code able to:

- cause some effect (e.g., send text to the terminal, create a file, draw an image, play a sound, etc.); this is something completely unheard of in the world of mathematics;
- evaluate a value (e.g., the square root of a value or the length of a given text) and return it as the function's result; this is what makes Python functions the relatives of mathematical concepts.

Moreover, many Python functions can do the above two things together.

Skill 0.1 Exercise 1

Skill 0.2: Interpret a built-in Python function

Skill 0.2 Concepts

We know that a function may have:

- an effect
- a result.

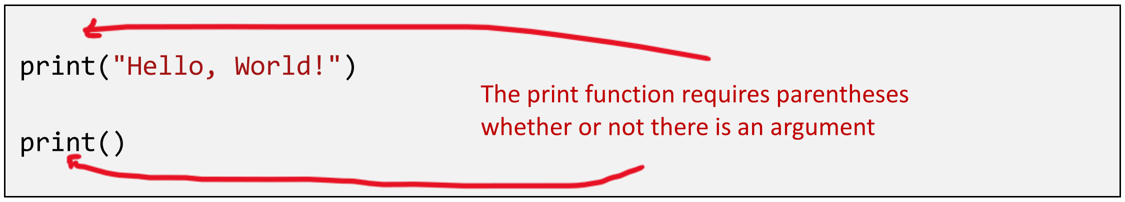
There's also a third, very important, function component – the argument(s).

Mathematical functions usually take one argument. For example, sin(x) takes an x, which is the measure of an angle.

Python functions, on the other hand, are more versatile. Depending on the individual needs, they may accept any number of arguments – as many as necessary to perform their tasks. Note: When we said *any number*, that includes zero – some Python functions don't need any argument.

Regardless of the needed/provided arguments, Python functions require the presence of **a pair of parentheses** – opening and closing ones, respectively.

If you want to deliver one or more arguments to a function, you place them **inside the parentheses**. If you're going to use a function which doesn't take any argument, you still have to have the parentheses.



```
print("Hello, World!")

print()
```

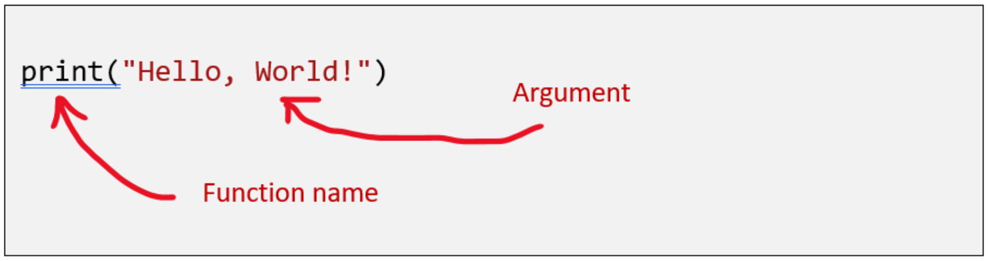
The print function requires parentheses whether or not there is an argument

[Skill 0.2 Exercise 1]

Skill 0.3: Explain the process of function invocation

Skill 0.3 Concepts

To understand function invocation let's consider what happens when Python encounters the code below,



```
print("Hello, World!")
```

- First, Python checks if the name specified is **legal** (it browses its internal data in order to find an existing function of the name; if this search fails, Python aborts the code)
- Second, Python checks if the function's requirements for the number of arguments **allows you to invoke** the function in this way (e.g., if a specific function demands exactly two arguments, any invocation delivering only one argument will be considered erroneous, and will abort the code's execution)
- Third, Python **leaves your code for a moment** and jumps into the function you want to invoke; of course, it takes your argument(s) too and passes it/them to the function;
- Fourth, the function **executes its code**, causes the desired effect (if any), evaluates the desired result(s) (if any) and finishes its task;
- Finally, Python **returns to your code** (to the place just after the invocation) and resumes its execution.

[Skill 0.3 Exercise 1]

Skill 0.4: Explore the print function

Skill 0.4 Concepts

The signature for the print function is shown below. According to the signature, the print function can accept many parameters or none at all.

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

We will begin our exploration of the print function by answering the following questions,

1. What effect does the *print()* function cause?

The effect is very useful and very spectacular. The print function:

- takes its arguments (it may accept more than one argument and may also accept less than one argument)
- converts them into human-readable form if needed (as you may suspect, strings don't require this action, as the string is already readable)
- and **sends the resulting data to the output device** (usually the console); in other words, anything you put into the *print()* function will appear on your screen.

No wonder then, that from now on, you'll utilize *print()* very intensively to see the results of your operations and evaluations.

2. What arguments does *print()* **expect?**

Any. We'll show you soon that *print()* is able to operate with virtually all types of data offered by Python. Strings, numbers, characters, logical values, objects – any of these may be successfully passed to *print().*

3. What value does the *print()* **function return?**

None. Its effect is enough.

A function invocation is one of many possible kinds of Python **instructions.** Of course, any complex program usually contains many more instructions than one.

Python's syntax is quite specific. Below lists some guidelines for writing instructions in Python.

- Unlike most programming languages, Python requires that **there cannot be more than one instruction in a line**. For example, the following would produce an error,

Code	Output
`print("Python is awesome!") round(5.2)`	SyntaxError: invalid syntax

- A line can be empty (i.e., it may contain no instruction at all)

Code	Output
`print("Python is awesome!")`	Python is awesome!
	5
`print(round(5.2))`	

- Python allows one instruction to spread across more than one line (which may be helpful when your code contains complex constructions).

Code	Output
`print("Python is awesome!",`	Python is awesome! Python is awesome! Python is awesome!
`    "Python is awesome!",`	
`    "Python is awesome!")`	

Now let's look at what happens when the print function is invoked without an argument. Notice, in the code below, that the empty print() invocation is not as empty as you may have expected – it outputs a new line.

Code	Output
`print("The itsy bitsy spider")`	The itsy bitsy spider
`print()`	
`print("climbed up the waterspout.")`	climbed up the waterspout.

Skill 0.4 Exercise

Skill 0.5: Apply the escape and new line characters

Skill 0.5 Concepts

We have seen that we can use an empty *print()* invocation to create a new line. This is not the only way to produce a **new line**. Now we are going to learn another way.

Below we've modified the code from above. Look at it carefully. Notice it produces the same output, but in a different way.

Code	Output
`print("The itsy bitsy spider\n\nclimbed up the waterspout.")`	The itsy bitsy spider
The \n can also be used to create a new line	climbed up the waterspout.

The backslash (\) has a very special function when used inside strings – this is called **the escape character**. The backslash serves as an announcement, that the next character after the backslash has a different meaning. In the example above, the letter **n** after the backslash means "new line".

There are other letters that can be used after the backslash. For example, a **t** after the backslash means "tab". We will learn more about escape characters in a few lessons.

Skill 0.5 Exercise 1

Skill 0.6: Use multiple arguments in a print function

Skill 0.6 Concepts

So far, we have tested the **print()** function behavior with no arguments, and with one argument. It's also worth trying to feed the print() function with more than one argument.

Look at the example below,

Code	Output

`print("The itsy bitsy spider",`
`"climbed up",`
`"the waterspout.")`
 	
The three arguments are separated by commas. The commas are part of the syntax and are not displayed when printed.
	The itsy bitsy spider climbed up the waterspout
	
	The arguments are printed on one line and there are spaces between each argument

In the example above, there is one *print()* function invocation, but it contains **three arguments**. All of them are strings. The arguments are **separated by commas**.

In this case, the commas separating the arguments play a completely different role than the comma inside the string. The former is a part of Python's syntax, while the latter is intended to be shown in the console.

Also, notice,

- The *print()* function invoked with more than one argument **outputs them all on one line**;
- The *print()* function **puts a space between the outputted arguments**.

Skill 0.6 Exercise 1

Skill 0.7: Use keyword arguments in the print function

Skill 0.7 Concepts

Python offers another mechanism for the passing of arguments, which can be helpful when you want to change the behavior of the print function. We aren't going to explain it in depth right now. For now, we will just explore how it works.

The mechanism is called **keyword arguments**. The *print()* function has two keyword arguments that you can use for your purposes. The first is called *end* and is illustrated below,

Code	Output

`print("My name is", "Python.", end=" ")`
`print("Monty Python.")`
The end key word consists of the key word, end, an equal sign and a value assignment. It must be the last argument in the print statement
	My name is Python. Monty Python.
	Assigning an empty space to the end keywork, overrides the default value (\n) and causes everything to print on one line

To use the end keyword, we have to use some basic rules,

- A keyword argument consists of three elements: a **keyword** identifying the argument (end here); an **equal sign** (=); and a **value** assigned to that argument;
- Any keyword arguments have to be put **after the last positional argument** (this is very important)

In the example above, we have made use of the end keyword argument and set it to a string containing one space. As you can see, the end keyword argument determines the characters the print() function sends to the output once it reaches the end of its positional arguments. The default behavior reflects the situation where the end keyword argument is **implicitly** used in the following way: end="\n".

The default separator for the arguments in the print function is a space. But, we can change this behavior with the *sep* keyword.

Code	Output

`print("My", "name", "is", "Monty",`
`"Python.", sep="-")`
	My-name-is-Monty-Python.

By assigning the keyword *sep* to a dash, the print() function uses a dash, instead of a space, to separate the outputted arguments.

Skill 0.7 Exercise 1