TASK

# The String Data Type

Visit our website

# Introduction

## WELCOME TO THE STRING DATA TYPE TASK!

Strings are some of the most important and useful data types in programming. Why? Let's think about it like this. When you were born, your parents did not immediately teach you to do mathematical sums, not 1 + 1 or standard deviation. The first thing they taught you to do was to speak, to say words, to construct full sentences, to say "Mom" or "Dad". Well this is why we are going to 'teach' the computer to first be able to communicate with the user — and the only way to do this is to have a good grasp of strings.

Get in touch

**Connect for support**

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

## WHAT ARE STRINGS?

A *string* is a list of letters, numerals, symbols and special characters that are put together. The values that we can store in a string are vast. An example of what strings can store is the surname, name, address, etc. of a person.

In Python, strings must be written within "quotation marks" for the computer to be able to read it.

The smallest possible string contains zero characters and is called an *empty string* (i.e. **`string = “”`**).

Examples of strings:

```
name = "Linda"

song = "The Bird Song",

licence_plate = "CTA 456 GP"
```

*Strings* are probably the most important data type in programming. They are used as a medium of communication between the computer and the user. The user can enter information as a string and the program can use the data to perform calculations and finally display the calculated answer to the user.

```
name = "John"
joke = "Knock, knock, Who's there?"
```

You can use any name for your variable but the actual string you are assigning to the variable must be within " " (quotation marks).

**Defining Multi-Line Strings:**
Sometimes, it's useful to have long strings that can go over one line. We use triple single quotes ( “”” “””) to define a multi-line string. Defining a multi-line string preserves the formatting of the string.

For example:

```
long_string = ''' This is a long string
 using triple quotes preserves everything inside it as a string
 even on different lines and with different \n spacing. '''
```

## STRING FORMATTING

Strings can be added to one another. In the past we used to use a method called **concatenation**, which looked like this:

```
name = "Peter"
surname = "Parker"
full_name = name + surname
```

*full_name* will now store the value “PeterParker”.

The + symbol simply joins the strings. If you wanted it to make your code more presentable, you could put spaces between the words.

```
full_name = name + " " + surname
```

We now added a blank space in between the two strings, so full_name will now store the value “Peter Parker”. **Now, you cannot concatenate a string and a non-string**. You need to cast the non-string to a string if you want to concatenate it with another string value. If you try to run code that adds a string and a non-string, you will get an error. For example, if we wanted to add an age of 32 we would have to cast it as a string to print it.

```
print(full_name + str(32))
```

**But this is a clunky way of formatting strings.** This way of putting together a string is still done in older languages, such as Java, and does have its place, but it is much better practice to use the *format()* method.

```
name = "Peter Parker"
age = 32
sentence = "My name is {} and I'm {} years old.".format(name, age)
print(sentence)
```

In the example above, a set of opening and closing curly braces ({}) serve as a placeholder for variables. The variables that will be put into those placeholders are listed in the brackets after the keyword *format*. The variables will fill in the placeholders in the order in which they are listed. Therefore, the code above will result in the following output: *My name is Peter Parker and I'm 32 years old.*

Notice that you don’t have to cast a variable that contains a number (*age*) to a string when you use the *format* method.

### f-Strings

The shorthand for the format function is *f-strings*. Take a look at the example below:

```
name = "Peter Parker"
age = 32
sentence = f"My name is {name} and I'm {age} years old."
print(sentence)
```

In f-strings, instead of writing *.format()* with the variables at the end, we write an *f* before the string and put the variable names within the curly brackets. This is a neat and concise way of formatting strings.

## NUMBERS AS STRINGS

We can even store numbers as strings. When we are storing a number ( i.e. 9, 10, 231) as a string, we are essentially storing it as a word. The number will lose all its number-defining characteristics. So the number will not be able to be used in any calculations. All you can do with it is read or display it.

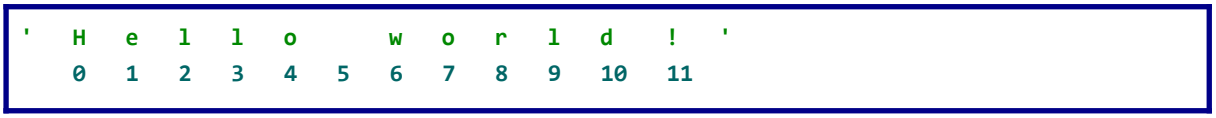
In real-life sometimes we don't need numbers to do calculations, we just need them for information purposes. For example, the house number you live in (let's say, 45 2nd Street JHB 2093) won't be used for performing any calculations. What benefit would there be for us to find out what the sum is of all the house numbers in an area? The only thing we need is to know what number the house is when visiting or delivering a package. The number just needs to be visible. This is the same concept when storing a number as a string; all we want is to be able to take in a value and display it to the user.

Example:

```
telephone_num = "041 1231234"
```

## THINGS YOU CAN DO TO STRINGS

A string is essentially a list of characters. For example, the word "Hello" is made up of the characters H+e+l+l+o. We can use this to our advantage to access the exact character we need using indexing.




Each character of a string (including blank spaces) is indexed by numbers starting from 0 for the first character on the left.

```
greeting = "Hello"
print(greeting[0] + greeting[1] + greeting[2] + greeting[3] + greeting[4])
```

**Output:**

```
Hello
```

You can use *len()* to get the number of characters in a string or length of a string. The print statement below prints out 12, because "Hello world!" is 12 characters long, including punctuation and spaces.

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

**Output:**

```
12
```

You can also *slice* a string. Slicing in Python extracts characters from a string, based on a starting index and ending index. It enables you to extract more than one character or "chunk" of characters from a string. The print statement below will print out a piece of the string. It will start at position/index 1, and end at position/index 4 (which is not included).

```
greeting = "Hello"
print(greeting[1:4])
```

**Output:**

```
ell
```

You can even put negative numbers inside the brackets. The characters are also indexed from right to left using negative numbers, where -1 is the rightmost index and so on. Using negative indices is an easy way of starting at the end of the string instead of the beginning. This way, -3 means "3rd character from the end".

Look at the example below. The string is printed from the first index, 'e', all the way to the end of the string. Notice that you do not need to specify the end of the index range:

```
greeting = "Hello"
print(greeting[1:])
```

**Output:**

```
ello
```

In the example below, the slice begins from position 0 and goes up to but not including position 1:

```
greeting = "Hello"
print(greeting[:1])
```

**Output:**

```
H
```

In the next example, the slice begins from position 1, includes positions 1 and 2, and then continues to the end of the string and skips/steps over every other position. This is known as an extended slice. The syntax for an extended slice is [begin : end : step]. If the end is left out, the slice continues to the end of the string.

```
greeting = "Hello"
print(greeting[1::2])
```

**Output:**

```
el
```

In this final example, you can think of the '-1' as a reverse order argument. The slice begins from position 4, continues to position 1 (not included) and skips/steps backwards one position at a time:

```
greeting = "Hello"
print(greeting[4:1:-1])
```

**Output:**

```
oll
```

You can print a string in reverse by using [::-1]. Remember that the syntax for an extended slice is [begin : end : step]. By not including a beginning and end and specifying a step of -1, the slice will cover the entire string, backwards, so the string will be reversed. You can find out more about extended slices **here**.

**Note** that slicing a string does not modify the original string. You can capture a slice from one variable in a separate variable. Try typing the following into the interactive shell:

```
new_string = "Hello world!"
fizz = new_string[0:5]
print(fizz)
print(new_string)
```

By slicing and storing the resulting substring in another variable, you can have both the whole string and the substring handy for quick, easy access.

## STRING HANDLING

There are various functions in Python that we can use to manipulate strings. Functions are used to save us from having to write monotonous code over and over. They have built-in code that is stored in Python.

Here are a few examples of useful string functions:

- The *upper()* and *lower()* functions make a new string with all letters converted to uppercase and lowercase, respectively.

```
a_string = "Hello World"

print(a_string.upper())      # prints out HELLO WORLD

print(a_string.lower())      # prints out hello world
```

- The *replace()* function will replace any occurrence of a string with a string of your choice. In the example below, every *$* character in the string stored in the variable *a_sentence* will be replaced with a space character.

```
a_sentence = "Welcome$to$the$world$of$programming"
```

```
print(a_sentence.replace("$" , " "))  # prints out Welcome to the world
of programming
```

- The *strip()* function is used to remove a certain character from the start and end of a string value. In the example below, the *strip()* function will remove all the * characters from the start and end of the string value stored in the variable named *str_help*.

```
str_help = "******Please leave me alone******"

print(str_help.strip('*'))  # prints out Please leave me alone
```

Most programming languages provide built-in functions to manipulate strings, i.e., you can concatenate strings, you can search from a string, you can extract substrings from a string, etc. We will cover string handling and more functions in a future task.

## ESCAPE CHARACTER

Python uses the backslash (\) as an escape character. The backslash (\) is used as a marker character to tell the compiler/interpreter that the next character has some special meaning. The backslash, together with certain other characters, is known as an 'escape sequence'.

Some useful escape sequences are listed below:

- **\n** - Newline
- **\t** - Tab
- **\s** - Space

The escape character can also be used if you need to include quotation marks within a string. You can put a backslash (\) in front of a quotation mark so that it doesn't terminate the string. You can also put a backslash in front of another backslash if you need to include a backslash in a string.

A note from our coding mentor

**Nkosi**

*Hey again, Have you heard about a Raspberry Pi before? No, it's not a new dessert, let me explain.*

***WHAT IS A RASPBERRY PI?***

*The Raspberry Pi is a computer that is so small it can fit in your pocket. You can plug it into your TV and connect a keyboard to it and you will have a fully setup personal computer. Don't be so quick to judge it just because of its size. It can be used for electronics projects, and for many of the things that your desktop PC does, like spreadsheets, word processing, browsing the internet, and playing games. It also plays high-definition video.*

*Imagine all that power and it is only as big as a credit card. This is the newest outlet for where your programs can be run on. It is always useful to know what new developments in technology are happening so that you don't fall behind.*

*A Raspberry Pi is relatively cheap to buy here in South Africa. If you want to learn more about it you can visit:* ***https://www.raspberrypi.org/***

# Instructions

Before you get started, we strongly suggest you start using an editor such as VS Code or Anaconda to open all text files (.txt) and Python files (.py).

First, read **example.py**, open it using an editor of your choice (VS Code, Anaconda).

- **example.py** should help you understand some simple Python. Every task will have example code to help you get started. Make sure you read all of **example.py** and try your best to understand.

- You may run **example.py** to see the output. Feel free to write and run your own example code before doing the Task to become more comfortable with Python.

# Compulsory Task 1

Follow these steps:

- Create a new Python file in this folder called **strings.py**
- Declare a variable called *hero* that contains the value "$$$Superman$$$"
- Use the string manipulation method `strip()` and print *hero* so the output is: Superman

## Compulsory Task 2

Follow these steps:

- Create a new Python file in this folder called **replace.py**.
- Save the sentence: “The!quick!brown!fox!jumps!over!the!lazy!dog!.” as a single string.
- Reprint this sentence as “The quick brown fox jumps over the lazy dog.” using the *replace()* function to replace every “!” exclamation mark with a blank space.
- Reprint that sentence as: “THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.” using the *upper()* function
- Print the sentence in reverse.

## Compulsory Task 3

Follow these steps:

- Create a new Python file in this folder called **manipulation.py**.
- Ask the user to enter a sentence using the *input()* method. Save the user’s response in a variable called *str_manip*.
- Using this string value, write the code to do the following:
    - Calculate and display the length of *str_manip*.
    - Find the last letter in *str_manip*. Replace every occurrence of this letter in *str_manip* with ‘@’.
        - e.g. if *str_manip* = “This is a bunch of words”, the output would be: “Thi@ i@ a bunch of word@”
    - Print the last 3 characters in *str_manip* backwards.
        - e.g. if *str_manip* = “This is a bunch of words”, the output would be: “sdr”.
    - Create a five-letter word that is made up of the first three characters and the last two characters in *str_manip*.

        - e.g. if *str_manip* = "**Thi**s is a bunch of wor**ds**", the output would be: "Thids".
    - Display each word on a new line.

If you are having any difficulties, please feel free to contact our specialist team **on Discord** for support.

## Optional Bonus Task

Let's practise some casting. Follow these steps:

- Create a new file called **optional_task.py**.
- Write Python code to take the name of a user's favourite restaurant and store it in a variable called *fav_rest*.
- Below this, write a statement to take in the user's favourite number. Use casting to store it in an integer variable called *int_fav*.
- Print out both of these using two separate print statements below what you have written. Be careful!
- Once this is working, try cast *fav_rest* to an integer. What happens? Add a comment in your code to explain why this is.

### Thing(s) to look out for:

1. Make sure that you have installed and setup all programs correctly. You have setup **Dropbox** correctly if you are reading this, but **Python or your editor** may not be installed correctly.
2. If you are not using Windows, please ask an expert code reviewer for alternative instructions.

Rate us

# Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

**Click here** to share your thoughts anonymously.