TASK

The String Data Type

Visit our website

Introduction

WELCOME TO THE STRING DATA TYPE TASK!

In the previous lesson, we learnt about different types of variables, including strings. Strings are useful because not only do they help us store all sorts of text and characters, they are also helpful when communicating information from the computer program to the user.

Get in touch

Connect for support

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

The best way to get help is to log in to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor 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 IS A STRING?

A **string** in Python is a sequence of characters. It can be a combination of letters (for instance, an entire sentence in English), numerals, symbols or special characters.

A string can be used to store any kind of information, such as a person's name or address, or a piece of text. The smallest possible string contains zero characters and is called an empty string.

Strings must be written within quotation marks (“ ” or ‘ ’) in order for the computer to be able to read it. In Python, single-quoted strings and double-quoted strings are the same, so ‘hello’ is the same as “hello”. However, if either a single or double quote is a part of the string itself, then you will need to let the Python interpreter know that that quote is not the end of the string. You do this by putting a \ before the quote that is meant to be part of the string. This slash is called an escape character, and won't show up when the string is printed. We will discuss the escape character in more detail later.

Some examples of strings are:

```
a_name = "Linda"

song = "The Bird Song is my favourite song."

licence_plate = "CTA 456 GP"

speech = "Then she said \"Hello there!\" "
```

We will now learn to perform some basic operations using strings.

BASIC STRING OPERATIONS

Concatenation of strings

Strings can be added to one another, which is called concatenation of strings.

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

`full_name` will now store the value "PeterParker". Note that there is no space between 'Peter' and 'Parker' in `full_name`, as that was not specified separately. The + symbol simply joins the strings as they are. A space character can be introduced between the given name and surname in the following manner:

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

The variable `full_name` will now store the value "Peter Parker" (not "PeterParker", as it did earlier). **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 *format()*, which we mentioned briefly in the previous task.

```
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.

Storing numbers as strings

We can even store numbers as strings. When we store a number (e.g. 9, 10, 231) as a string, we are essentially storing it as a text. The number will lose all its number defining characteristics, and can no longer function as a numeral in mathematical calculations. All you can do with a number stored as text (in the form of a string) is read or display it.

You may be wondering why you would need to do this at all. Sometimes, you don't need numbers to actually do calculations. You just need them for the purpose of storing or displaying information. For example, your street address may be 45 Rose Street Tokyo 116-0002. If you input this into a computer program, you won't be performing any calculations with this address. You simply want the computer to be able to take in this value, store it, and display it to you (or another user) whenever needed. Storing this address as a string would meet all those requirements.

INDEXING STRINGS

Accessing elements of a string

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 are able to use this to our advantage as we can access specific characters from a string using indexing.

Think of a string as an ordered list of characters, with each character in the string indexed to its position within the string (expressed as a number). Each character of a string, including blank spaces, is indexed by a number starting from 0 for the first character on the left.

We can use square brackets to access elements of the string, with the number within the square brackets signifying the position of the character to be accessed.

What, then, would the following program output?

```
my_string = "Hello world!"
char = my_string[2]
print(char)
```

If your answer is ‘e’ because that is the second character in the string *Hello world!* - think again! The correct answer is ‘l’. This is because the beginning character of a string corresponds to index 0, not 1. This means the characters in the string *Hello world!* are indexed as shown below:

H	e	l	l	o		w	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11

The space and exclamation point are included in the character count, so *Hello world!* is 12 characters long, from ‘H’ at index 0 to ’!’ at index 11. Remember that if you specify an index, you’ll access the character at that position in the string, as shown below:
(Note: The character after each print function is the output of that command)

```
string = "Hello"
print(string[0])
#Output: H
print(string[1])
#Output: e
print(string[4])
#Output: o
```

Characters in a string 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, which is helpful if your string is really long. This way, the index -3 means "third character from the end". Refer to the file **example.py** within this task folder to see the output for a sample program.

Determining string length

You can use `len()` to get the number of characters in a string, that is, the 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
```

Slicing strings

You can also slice strings. Slicing extracts characters from a string based on a starting index and ending index. This enables you to extract more than one character or a chunk of characters from a string.

Keep in mind that when doing so, the starting index is included in the output while the ending index is not.

Try typing the following into the Python shell:

```
new_string = 'Hello world!'
fizz = new_string[0:5]
print(fizz)
#Output: Hello
```

The print function below will print out a piece of the string, starting at position/index 3, and end at position/index 6.

```
new_string = 'Hello world!'
print(new_string[3:8])
#Output: lo wo
```

In the above example, notice how the ending index character is not included when returning the output (the sliced extract ends at index number 7, as the range is specified until index number 8).

Also, note that slicing a string does not modify the original string. If you wish, you can store a slice from one variable in a separate variable. 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 Methods

Once you understand strings and their indexing, the next step is to master using some common string methods. These are built-in modules of code that perform certain operations on strings. These methods are really useful as they save time since there is no need to write the code over and over again in order to perform certain operations.

The most common of these methods are provided below.

For any string variable defined as:

`str = “Oh hello world!”`

- `str.lower()`
  This converts the string to lowercase.

- `str.upper()`
  This converts the string to uppercase.

- `str.strip()`
  This removes any whitespaces from the beginning or end of the string. An optional argument can be supplied to strip any other character other than space, in which case this method can be used as follows **`str.strip(char).`**

- `str.find(‘text’)`
  This searches the given string for a specific substring (‘text’) and returns the lowest index of the substring if it is found. If the substring is not found, then this method returns -1. This method can also contain optional arguments that can specify the start and endpoint of the range within which to search. In that case, the method would be written as **`str.find(‘text’, start, end)`** where ‘start’ and ‘end’ are numbers denoting the index of the range within which to search. When using optional arguments, remember that if the substring is in fact found within the delimited range, the index returned will correspond to the original string (i.e. the delimited range will not be treated as a new string).

- `str.replace('oldText', 'newText')`
  This replaces any occurrence of **`‘oldText’`** with **`‘newText’`** but will not overwrite the original string variable, so should be assigned to a new variable if you wish to use the new string.

- `str.split('word')`
  This breaks down a string into a list of smaller pieces. The string is separated based on what is called a delimiter. The delimiter can be a string or a character like a comma or whitespace. If no delimiter is specified, this method will automatically split the string using whitespace as the delimiter.

- `str.count(‘text’)`
  This returns the number of occurrences of the substring **`‘text’`** in the given string. This method can also contain optional arguments that can specify the start and endpoint of the range within which to search. In that case, the method would be written as **`str.count(‘text’, start, end)`** where **`‘start’`** and **`‘end’`** are numbers denoting the index of the range within which to search.

Make sure to go through 'example.py' in this folder to see how each of these methods can be used.

ESCAPE CHARACTER

Python uses the backslash (\) as an escape character.

The backslash is used as a marker character to inform the 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, as discussed earlier. 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.

Instructions

Before you get started we strongly suggest you start using Notepad++ or IDLE to open all text files (.txt) and Python files (.py). Do not use the normal Windows Notepad or Mac TextEdit as it will be much harder to read.

First, read *example.py* in this task folder by opening it with your IDLE or Notepad++ (right-click the file and select 'Edit with Notepad++').

- The file 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.
- Do run example.py to see the output. Feel free to write and run your own example code before doing the tasks to become more comfortable with Python.
- This may take some time at first as it is different from other (natural) languages, but persevere! Your mentor is here to assist you along the way.

Compulsory Task 1

- Create a new Python file in this folder called **strings.py**
- Declare the variable called hero = “Super Man”
- Print it out in the following way:
    - S^U^P^E^R M^A^N

Compulsory Task 2

- 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.
- Now reprint this sentence as “The quick brown fox jumps over the lazy dog” using the `replace()` function to change every “!” exclamation mark into a blank space.
- Now 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

- Create a new file called **cast.py**.
- Write Python code to take the name of a user's favourite restaurant and store it in a variable called `fav_rest`.
- Now, below this, write a statement to take in the user's favourite number. Use casting to store it in a variable called `int_fav`.
- Now 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 int. What happens? Add a comment in your code to explain why this is.

Completed the task(s)?

Ask your mentor to review your work!

Review work

Thing(s) to look out for:

Make sure that you have installed and set up **Dropbox** correctly on your machine.

Rate us

Share your thoughts

HyperionDev 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.