Introduction to Computer Science

Lesson 1.0: String Basics

Objectives:

> - Create strings in Python.
> - Use variables to represent strings.

Creating Strings

Strings or **string literals** are a sequence or characters. They are usually used to represent human language, file names, web addresses, etc. Things that are non-numeric sequences of characters are usually strings, but it is possible for a sequence of numeric characters to be a string as well. Stings in Python are created with either single quotes (`' '`) or double quotes (`" "`). You will definitely see them both ways. Some people prefer to use single quotes as it is one less key stroke. Here are some examples of strings:

- `'We are leaning Python'`
- `"Python is the 3rd most popular programming language."`
- `'a'`
- `'main.py'`
- `"https://www.google.com/"`
- `'3.14159265358979323846264338327 9'`
- `" "`

Notice how the examples above are a sequence of characters between either `' '` or `" "`. The first two examples are sentences. The third is a single character. The fourth example is a file name. The fifth is the URL to Google. And even though the sixth example may appear to be numeric, it is also a string. The final string example is just a space. Even though this may not seem like a character, Python recognizes it as a character. This is an example of what is known as **whitespace**. Whitespace will be covered more in the subsequent lessons.

As mentioned previously, it does not matter if `' '` is used or `" "` are used. Python will interpret them the same way. In fact it does not affect the string between the two types of quotes mention. We can test two strings using the **equality operator** (`==`) to determine if the they are the same or equal. Try the code below **Example 1** in the REPL:

Example 1

```
'Python' == "Python"
```

The output is as follows:

```
True
```

The interpreter will output `True`, because the strings are identical between the two different quotes. Now try to input the following code below **Example 2** in the REPL:

Example 2

```
'Python' == "python"
```

The output is as follows:

```
False
```

The interpreter will output `False`, because the first string has a capital `P` and the second string has a lowercase `p`. So even though both strings say 'Python', Python can determine that there is a difference.

Variables

When you create an object such as a string in Python, it is stored somewhere on the memory. **Variables** give you a way to access the string so it can be used again. You can create variables that point to strings using the assignment operator `=`. Variables are created as follows:

```
first_name = 'Bob'
```

You can now print the name `Bob` using the variable instead of the string. Try the code following **Example 3** in the REPL.

Example 3

```
first_name = 'Bob'
print(name)
```

The output is as follows:

```
Bob
```

Choosing Variable Names

Choosing variable names has limitations:

Limitation	Good	Bad
Variables cannot begin with a number.	`name1`	`1name`
Variables cannot have special characters other than an underscore (`_`).	`first_name`	`first-name`
Variables cannot have whitespace.	`favorite_food`, `favoriteFood`	`favorite food`

In addition to the limitations above, variables cannot be named using one of the **reserve words** or **keywords** in Python. These words have a special meaning in Python. Here is a list of these words:

```
and           continue      finally       is            raise
as            def           for           lamda         return
assert        del           from          None          True
async         elif          global        nonlocal      try
await         else          if            not           while
break         except        import        or            with
class         False         in            pass          yield
```

To see the list of keywords type the following two commands in the interpreter.

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break',
'class', 'continue', 'def',
'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if',
'import', 'in', 'is', 'lambda',
'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with',
'yield']
```

There are also a couple of things that should be avoided for a more pythonic style. The following is allowed but are typically used for purposes that are beyond the scope of this lesson.

- Avoid using all capitals. (i.e. `FIRST_NAME`)
- Avoid using a capital letter at the beginning. (i.e. `Favorite_food`)

Lab 1.0: String Basics

Write a program that describes AND prints five things about yourself using five variables using five `print()` functions.

Here is an example program:

```
 # Declare FIVE variables with strings that describe yourself.
name = 'My name is Simon Escalada-Mastick.'
degrees = 'I studied linguistics and mathematics in college.'
occupation = 'I am a data analyst and teacher at HPLS.'
city = 'I live in Tucson, Arizona.'
favorite_restaurant = 'My favorite restaurant in Tucson is Yoshimatsu.'

# Print the five variables.
print(name)
print(degrees)
print(occupation)
print(city)
print(favorite_restaurant)
```

Turn in the `string_basics.py` or the `string_basics.ipynb` file in Google Classroom for full credit.