Day 3

Strings

Objectives

Students should be able to:

- Concatenate strings.
- Index strings.
- Slice Strings.
- Convert strings into other types.
- Formatting other types into strings.

Strings

- As we have seen previously, we can assign strings into variables, print them out, and convert them into integers or floats using the int and float functions.
- There are also many other things we can do with strings.
- Such as get a particular character (or slice of characters) from a string.
- Or concatenate (combine) several strings together.
- We can also convert other types into strings so that we can display more aesthetic messages.

String Indexing

String Indexing

- A string is simply a sequence of characters.
- For example, the string ‘Hi 12’ is made up of 5 characters: the letters ‘H’ and ‘i’, a space, and the digits ‘1’ and ‘2’.
- We can extract a particular character from a string once we know its position in the string.
- This is done using indexing.

Example

Source Code

```
my_name = 'John Doe'
print(my_name[3])
```

Output

```
n
>>>
```

Explanation

- We have a variable called my_name assigned to the string ‘John Doe’.
- We then print out my_name[3], only a single character gets printed out. Specifically, the letter ‘n’.
- The square brackets is called the indexing operator.
- It is used to extract a character from a string.
- The integer inside the brackets specifies which character to extract. The 3 inside the brackets means that we want to extract the 4^{th} character, which is the ‘n’.

Example

Source Code

```
my_name = 'John Doe'
print(my_name[3])
```

Output

```
n
>>>
```

Explanation continued

- You might ask why use the number 3 to get the 4th character.
- This is because the index numbering start at 0. So, the ‘J’ is at index 0, the ‘o’ is at index 1, and so on.

Example 2

Source Code

```
my_name = 'John Doe'
print(my_name[0])
print(my_name[1])
print(my_name[2])
print(my_name[4])
print(my_name[7])
```

Output

```
J
o
h

e
```

Explanation

- This example shows that ‘J’ is at index 0, ‘o’ is at index 1, ‘h’ at index 2.
- It also shows that the space is at index 4, proving that spaces are counted as part of the string.
- It also shows that the last character, the letter ‘e’ is at index 7, despite there being 8 characters in the string. This is, of course, because the indices start at 0.

Example 3

Source Code

```
message = 'Hi 12'
x = message[0]
print(x)
y = int(message[3])
z = int(message[4])
print(y+z)
print('Hello'[4])
```

Output

```
H
3
o
```

Explanation

- These examples show more diverse ways of using indexing.
- In lines 2 and 3, we set x to the character at index 0, which is the ‘H’, so when x is printed out the letter ‘H’ is displayed.
- On line 4, we extract the digit ‘1’ from the string. We then convert it into an integer and assign it to the variable y. Similarly, the variable z gets assigned to the integer 2 after extracting the digit from the string.

Example 3

Source Code

```
message = 'Hi 12'
x = message[0]
print(x)
y = int(message[3])
z = int(message[4])
print(y+z)
print('Hello'[4])
```

Output

```
H
3
o
```

Explanation Continued

- We then print out the sum of y and z, which are 1 and 2 respectively, so 3 is displayed.
- We can also index a string directly. So, on line 7, the character at index 4 from the string ‘Hello’, which is the letter ‘o’, is extracted, and hence printed out.

Example 4

Source Code

```
message = 'Congratulations!'
x = 7
print(message[x])
print(message[x+4])
```

Output

```
u
i
```

Explanation

- From this example, we see that we don't have to put an exact number as the index.
- We can put a variable instead.
- Or any calculation or expression which gives an integer as the answer.

Example 5 – Negative Indices

Source Code

```
message1 = 'Hi'
message2 = 'Hello'
message3 = 'Congratulations!'
print(message1[-1])
print(message2[-1])
print(message3[-1])
```

Output

```
i
o
!
```

Explanation

- Python also allows you to use negative indices to get characters starting from the end of a string.
- Notice that using an index of -1 always extracts the last character of the string.
- This does not work in most other programming languages, however.

Example 6 – Negative Indices

Source Code

```
msg = 'Congratulations!'
print(msg[-1])
print(msg[-2])
print(msg[-3])
print(msg[-16])
```

Output

```
!
s
n
C
```

Explanation

- Python also allows you to use negative indices to get characters starting from the end of a string.
- The indices -1, -2, and -3 correspond to the last, second to last, and third to last characters respectively, which are ‘!’, ‘s’, and ‘n’.
- This pattern continues until the first character, which in this case has an index of -16.
- This does not work in most other programming languages, however.

Index Errors

- Trying to use an index on a string which does not have enough characters to get to that index will cause the program to give an error.
- Using non-integers as indices also gives an error.

String Slicing

String Slicing

- String slicing is very similar to string indexing.
- But instead of extracting a single character from a string, it extracts a smaller string from the original string.

Example 1

Source Code

```
msg = 'Congratulations!'
print(msg[3:7])
print(msg[6:])
print(msg[:9])
```

Output

```
grat
tulations!
Congratul
```

Explanation

- Using two integers in the brackets separated by a colon allows you to extract a smaller string using a range of characters from the original string.
- Here we see that the slice [3:7] extracted the string “grat” from the original string “Congratulations!”.
- The 3 represents the index to start extracting at (‘g’ is at index 3)
- The 7 represents the index to stop extracting at.
- Notice that the character at index 7 (the ‘u’) is not included, That is because extracting a slice does not include the stopping index, it stops at the one before that.

Example 1

Source Code

```
msg = 'Congratulations!'
print(msg[3:7])
print(msg[6:])
print(msg[:9])
```

Output

```
grat
tulations!
Congratul
```

Explanation

- If the stopping index is left out, like on line 3, then the extraction will include the entire remainder of the string.
- If the starting index is left out, like on line 4, then the extraction will start from the first character.

Example 2 – Step Sizes

Source Code

```
msg = 'Congratulations!'
print(msg[2:14:2])
print(msg[1::3])
print(msg[:15:2])
```

Output

```
nrtlto
orutn
Cnrtltos
```

Explanation

- If a third integer is included in the brackets after a second colon, then that integer is the step size.
- If the step size is 2 (like on lines 2 and 4), then every other character is extracted.
- If the step size is 3 (like on line 3), then every third character is extracted.
- Etc.
- Again, the starting and stopping indices can be left out like on lines 4 and 3 respectively.

String Concatenation

String Concatenation

- String concatenation refers to joining several string together.
- In python this is done using the addition symbol.

Example 1

Source Code

```
first = input('Enter your first name: ')
last = input('Enter your last name: ')
print('Hello' + first + last)
print('Hello ' + first + ' ' + last)
print('Hello', first, last)
```

Output

```
Enter your first name: John
Enter your last name: Doe
HelloJohnDoe
Hello John Doe
Hello John Doe
```

Example 1

Explanation

- We ask the user to enter their first name and last name.
- In line 3 we then concatenate the string “Hello” with their first name and last name and print out the combined string.
- Notice, however, that it prints out all the strings stuck together.
- We can fix this by placing a space in the first string and adding a string with a space between the first and last name. This is done on line 4.
- In this example, we could have gotten the same result by using commas in the print function as in line 5.

Example 2 – Caution

Source Code

```
s1 = input('Enter your 1st salary: ')
s2 = input('Enter your 2nd salary: ')
total = s1 + s2
print(total)
```

Output

```
Enter your 1st salary: 1000
Enter your 2nd salary: 2000
10002000
```

Explanation

- Because the inputs were not converted into integers (or floats), when we tried to add them, they were still treated as string, and therefore were concatenated instead.

Example 3 – Incrementing

Source Code

```
x = 'Hello'
x = x + ' '
x = x + 'World'
x = x + '!'
print(x)
```

Output

```
Hello World!
```

Explanation

- Similarly, to how a numeric variable can be increased by adding a value to it and assigning it back into a variable,
- The equivalent can be done to strings.
- In this example, x starts off with the string “Hello”, and more strings are added to it until it contains the entire “Hello World!” string, which is then printed out.

String Formatting

String Formatting

- String formatting involves placing data (such as numbers) into a string
- This is normally done to display the data in a nice way for the user.
- There are many ways to format strings in Python.
- Four of these ways are shown in the following example.

Example 1 – Code

```
name = input('Enter your name: ')
age = int(input('Enter your age: '))
height = float(input('Enter your height in metres: '))
print('Name: %s Age: %d Height: %f' % (name, age, height))
print('Name: ' + name + ' Age: ' + str(age) + ' Height: ' + str(height))
print('Name: {} Age: {} Height: {}'.format(name, age, height))
print(f'Name: {name} Age: {age} Height: {height}')
```

Example 1 – Output

```
Enter your name: John Doe
Enter your age: 32
Enter your height in metres: 1.9876
Name: John Doe Age: 32 Height: 1.987600
Name: John Doe Age: 32 Height: 1.9876
Name: John Doe Age: 32 Height: 1.9876
Name: John Doe Age: 32 Height: 1.9876
```

Example 1 – Explanation of Method 1

- We first ask the user to enter their name, age and height.
- The age is converted into an integer and the height is converted into a float.
- The first method, on line 4, uses placeholders.
- The ‘%s’, ‘%d’, and ‘%f’ are known as placeholder.
- These placeholders will eventually be replaced with actual data.
- ‘%s’ is used as a placeholder for a string, ‘%d’ for an integer, and ‘%f’ for a float.
- Which is why they are used for the name, age, and height, respectively.
- To replace the placeholders, we type the % sign after the string and, in parentheses, list out the variables we want to replace the placeholders with.

Example 1 – Explanation of Method 2

- The second method, on line 5, uses string concatenation, and the str function.
- The str function is used to convert integers, floats and other data into a string.
- By using the str function on the name and the height, we can then concatenate all the pieces of data into a single string.

Example 1 – Explanation of Method 3 and 4

- The third method, on line 6, uses placeholders and the .format method.
- For this method, the placeholders are curly brackets.
- To replace the placeholders, we type .format() after the string and list out the variables we want to replace the placeholders with, in the parentheses of the format function.
- In the last method, on line 7, we do something very similar to method 3, but we place an f before the quotation mark, and we don’t use the format function.
- We also put the variable name in the curly brackets in the string instead of outside the string.

Formatting Floats

- Normally, when a float is printed out, many decimal places are displayed.
- We usually only want to print a few decimal places.
- This can be specified when printing out floats using formatting.

Example 2 – Code

```
height = float(input('Enter your height in metres: '))
print('Height: %.2f' % (height))
print('Height: ' + str(round(height, 2)))
print('Height: {:.2f}'.format(height))
print(f'Height: {height:.2f}')
```

Example 2 – Output

```
Enter your height in metres: 1.9876
Height: 1.99
Height: 1.99
Height: 1.99
Height: 1.99
```

Example 2 – Explanation

- This example shows how to specify the number of decimal places to print a float for each of the four formatting methods. In each case the number of decimal places was specified as 2.
- For the first method, the placeholder used was “%.2f” instead of “%f”. For the third and fourth methods, the specifier “:.2f” was added inside the curly brackets.
- For the second method, the round function was used. By placing the number 2 into the round function, the float is rounded to 2 decimal places.
- For each of these, the 2 can be changed to whatever number of decimal places you want.