Python Programming

Practical Questions - Database Management

Session 1

STRINGS

A **string** is an object that contains a sequence of characters. We can declare a Python string using a single quote, a double quote, a triple quote, or the `str()` function.

```
# A single quote string
single_quote = 'a'  # This is an example of a character in other programming
languages. It is a string in Python

# Another single quote string
another_single_quote = 'Programming teaches you patience.'

# A double quote string
double_quote = "aa"

# Another double-quote string
another_double_quote = "It is impossible until it is done!"

# A triple quote string
triple_quote = '''aaa'''

# Also a triple quote string
another_triple_quote = """Welcome to the Python programming language. Ready, 1, 2,
3, Go!"""

# Using the str() function
string_function = str(123.45)  # str() converts float data type to string data type

# Another str() function
another_string_function = str(True)  # str() converts a boolean data type to string
data type

# An empty string
empty_string = ''

# Also an empty string
second_empty_string = ""

# We are not done yet
third_empty_string = """""  # This is also an empty string: ''''''
```

Escape Characters in Python

Code	Result	Try it
\'	Single Quote	Try it »
\\	Backslash	Try it »
\n	New Line	Try it »
\r	Carriage Return	Try it »
\t	Tab	Try it »
\b	Backspace	Try it »

Another way of getting strings in Python is using the `input()` function. The `input()` function allows us to insert values into a program with the keyboard. The inserted values are read as a string, but we can convert them into other data types:

```
# Inputs into a Python program
input_float = input()  # Type in: 3.142
input_boolean = input() # Type in: True

# Convert inputs into other data types
convert_float = float(input_float)  # converts the string data type to a float
convert_boolean = bool(input_boolean) # converts the string data type to a bool
```

We use the `type()` function to determine the data type of an object in Python. It returns the class of the object. When the object is a string, it returns the `str` class. Similarly, it returns `dict`, `int`, `float`, `tuple`, `bool` class when the object is a dictionary, integer, float, tuple, or Boolean, respectively. Let's now use the `type()` function to determine the data types of variables declared in the previous code snippets:

```
# Data types/ classes with type()

print(type(single_quote))
print(type(another_triple_quote))
print(type(empty_string))

print(type(input_float))
print(type(input_boolean))

print(type(convert_float))
print(type(convert_boolean))
```

String Properties

Zero Index: The first element in a string has an index of zero, while the last element has an index of `len(string) - 1`. For example:

```
immutable_string = "Accountability"

print(len(immutable_string))
print(immutable_string.index('A'))
print(immutable_string.index('y'))
```

```
14

0

13
```

Immutability. This means that we cannot update the characters in a string. For example, we cannot delete an element from a string or try to assign a new element at any of its index positions. If we try to update the string, it throws a `TypeError`:

```
immutable_string = "Accountability"

# Assign a new element at index 0
immutable_string[0] = 'B'
```

We can, however, reassign a string to the `immutable_string` variable, but we should note that they aren't the same string because they don't point to the same object in memory. Python doesn't update the old string object; it creates a new one

Concatenation: joining two or more strings together to get a new string with the `+` symbol. For example:

```
first_string = "Data"
second_string = "quest"
third_string = "Data Science Path"

fourth_string = first_string + second_string
print(fourth_string)

fifth_string = fourth_string + " " + third_string
print(fifth_string)
```

```
Dataquest

Dataquest Data Science Path
```

Repetition: A string can be repeated with the `*` symbol. For example:

```
print("Ha" * 3)
```

```
HaHaHa
```

Indexing and Slicing: we already established that strings are zero-indexed. We can access any element in a string with its index value. We can also take subsets of a string by slicing between two index values. For example:

```
main_string = "I learned R and Python on Dataquest. You can do it too!"

# Index 0
print(main_string[0])

# Index 1
print(main_string[1])

# Check if Index 1 is whitespace
print(main_string[1].isspace())

# Slicing 1
print(main_string[0:11])

# Slicing 2:
print(main_string[-18:])

# Slicing and concatenation
print(main_string[0:11] + ". " + main_string[-18:])
```

```
I

True
I learned R
You can do it too!
I learned R. You can do it too!
```

String Operations

Looping through a string. Strings are iterable. Therefore, they support the looping operations with `for loop` and `enumerate`:

```
# For-loop example
word = "bank"
for letter in word:
    print(letter)
```

```
b
a
n
k
# Enumerate example
for idx, value in enumerate(word):
    print(idx, value)
```

```
0 b
1 a
2 n
3 k
```

String and relational operators: when two strings are compared using relational operators (`>`, `<`, `==`, etc.), the elements of the two strings are compared by their ASCII decimal numbers index by index.

Question 1

```
x = "Hello"
print(x)
```

Ans:

Question 2

```
print(x[0])
print(x[1])
```

Ans:

Question 3

```
x = "hello world"
s = x[0:3]
print(s)
s = x[:3]
print(s)
```

Ans:

Question 4 ******** **********

Create a String (give an appropriate name) that takes in User's First name and stores it. Create another String that takes in the User's Surname on a new line.

Finally, the final output should output a welcome message to the user on the screen

"*Welcome _ _ _ _ to CCDM Course*" on a new line.