Strings

Representing text data in Python

Alwin Tareen

The Definition of a String

- A string is how a computer represents text data.
- In `Python`, the data type of a string is: `str`
- Generally, a string is a sequence of characters.
- Each character is represented in `Python`'s memory as an `ASCII` value.
- For example, `"H"` is represented by the `ASCII` value `72`.
- Strings must be enclosed by single quotation marks, or double quotation marks.

Manipulating Strings as Sequences of Characters

- In order to manipulate a string, we need to be able to access the individual characters that make up a string.
- In `Python`, a string can actually be regarded as an array-like structure of elements. This allows us to access the internal parts of the string.
- If we wish to extract single characters from a string, we can use square bracket notation, along with a number called an **index**: `word[index]`

String Indexes

- Programming languages usually have a convention whereby the first element in a sequence is index `0`. Consider the following string:

```
fruit = "watermelon"
```

- The indexes of this string are as follows:

letter	w	a	t	e	r	m	e	l	o	n
index	0	1	2	3	4	5	6	7	8	9

- If we want to get the first letter of this string, we can use index `0`:

```
first = fruit[0]
```

String Indexes

- You can use any expression, including variables and operators as an index. However, the index must be an integer.
- In other words, it wouldn't make any sense to have a fractional value as an index: `fruit[1.5]`
- Another interesting aspect about indexes is that they can take on negative values. This means that the letters in the string have a wraparound effect, where the last letter can be easily accessed.

String Indexes

- The string `"watermelon"` can be regarded as follows:

letter	w	a	t	e	r	m	e	l	o	n
index	0	1	2	3	4	5	6	7	8	9
index	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- The last letter can be accessed with: `fruit[-1]`

The Quantity of Characters in a String

- We can use the built-in function `len()` to find out how many characters there are in a string:

```
vegetable = "cauliflower"
quantity = len(vegetable)
print(quantity)
```

- Another way to access the last character in a string is to provide an index where `1` is subtracted from the number of characters:

```
last = vegetable[quantity-1]
print(last)
```

String Slicing

- Sometimes, we want to extract several characters from a string.
- Slicing out some part of a string gives you a **substring**.
- For example, the strings `"eat"` and `"ted"` are substrings of `"repeated"`.
- We can use a variation of square bracket notation to extract a substring.
- However, we need two indexes, separated by a colon.
- The first index corresponds to the first letter that you **want**.
- The second index corresponds to the first letter that you **don't want**.

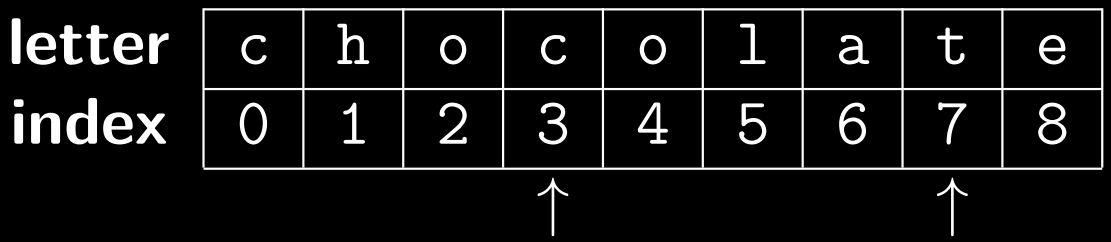
```
variablename[firstindex:lastindex]
```

String Slicing

- Consider the following string:

```
dessert = "chocolate"
```

- Let's say that we wanted to extract the string `"cola"`.
- Start with the index of the first letter that you want, 3.
- End with the index of the first letter that you don't want, 7.




```
drink = dessert[3:7]
print(drink)
```

String Slicing Shorthand

- `Python` allows you to use a special kind of shorthand notation with string slicing.
- For example, consider the following string:

```
breakfast = "pineapple"
```

- Say that you wanted to extract the word `"pine"`. You could do the following:

```
tree = breakfast[0:4]
```

- However, if you wanted your slice to start from the beginning of the string, you could omit the first index:

```
cone = breakfast[:4]
```

String Slicing Shorthand

- Similarly, if you wanted your slice to go all the way to the end of the string, then you could omit the second index as follows:

```
flavor = "strawberry"
```

- If you wanted to extract the word `"berry"`, then you may do the following:

```
snack = flavor[5:10]
food = flavor[5:]
```

String Slicing Shorthand

- What would happen if we performed a string slice in which both the first and last indexes were omitted?
- This would result in a **clone** of the string.

```
icecream = flavor[:]
print(icecream)
```

String Concatenation with the + Operator

- The + operator takes on a different role when its operands have the string data type.
- It joins the two strings, and returns the newly joined string.
- In other words, it creates a new string that starts with the first operand, and has the second operand immediately after it.

```
candy = "bubble" + "gum"
print(candy)
```

String Duplication with the * Operator

- The * operator takes on a different role when one of its operands is a string, and the other one is an integer.
- If a string is multiplied by an integer `n`, the result is a new string which has `n` copies of the original string, one after the other.

```
greetings = "hello" * 3
print(greetings)
```

Strings are Immutable

- Once a string has been created, it cannot be changed or altered.
- In other words, you cannot do the following:

```
grocery = "Mango"
grocery[0] = "T" # error
```

- If you insist on performing this action, then the best that you can do is to make a new string using part of the original string.

```
grocery = "Mango"
dance = "T" + grocery[1:]
print(dance)
```

Looping across a String

- Python regards a string as a series of elements.
- Therefore, we can loop across a string just like any other sequential data structure.
- The following is a traversal across a string using a `for` loop:

```
fruit = "raspberry"
for letter in fruit:
    print(letter)
```

- Each time through this `for` loop, the next character in the string is assigned to the variable `letter`.
- The loop continues until there are no more characters left.

Counting Items in a String

- We can use the `for` loop construct to count how many times a particular element is present in a string:

```
word = "banana"
count = 0
for letter in word:
    if letter == "a":
        count += 1
print(count)
```

The `in` Operator

- The `in` operator is used to determine if a substring appears in a given string.
- It returns `True` if the substring is contained in the given string, and `False` otherwise:

```
result = "a" in "banana"
print(result)
```

- The following is another example:

```
outcome = "raw" in "strawberry"
print(outcome)
```

The `not in` Operator

- You can combine the `not` operator with the `in` operator to determine if a string does **not** contain something:

```
snacks = "pizza, wings, burgers"
if "chips" not in snacks:
    print("You forgot the chips!")
```

Numerical ASCII Codes for Characters

- Every character symbol on your keyboard is actually represented internally in the computer by a numerical ASCII code.
- Generally, all modern computers use a standard set of characters which are represented by the numbers between 32 and 255.
- The following are some characters and their respective numerical codes:

Digits

Value	Symbol
48	0
49	1
50	2
51	3

Uppercase

Value	Symbol
65	A
66	B
67	C
68	D

Lowercase

Value	Symbol
97	a
98	b
99	c
100	d

Converting between Letters and ASCII Codes

The ord() function

- You can convert a character into its corresponding numerical ASCII code using the ord() function:

```
numcode = ord("P")
print(numcode)
```

The chr() function

- The chr() function performs the reverse operation: it takes in a numerical ASCII code as input, and returns the character corresponding to that code.

```
initial = chr(84)
print(initial)
```

String Comparison

- The comparison operators(==, <, >, etc.) work on strings.
- To check if two strings are equal, use the equality operator: ==

```
if password == "basketball"
    print("login successful")
```

- Since each character is associated with a particular numerical ASCII code, we can reason the following:

```
dgits < uppercase letters < lowercase letters
```

- This means that we can use the less than/greater than signs to compare strings:

```
if "999" < "thousand":
    print("access granted")
```

String Methods

- Calling a method is very similar to calling a function. Methods can take arguments and return values.
- However, the syntax is different.
- You call a method by using dot notation as follows:

```
objectname.methodname(arguments)
```

- We are going to examine specific methods that can be applied to strings.
- These are methods that return information about the string, or return a new string that is a modified version of the original.

String Methods: Letter Case

`capitalize()`

This returns a string in which the first character is upper case, and the rest of the string is lower case.

`lowercase()`

This returns a string with every letter of the original in lowercase.

`upper()`

This returns a string with every letter of the original in uppercase.

String Methods: Characters

`isalpha()`

This returns `True` is every character of the string is a letter, and `False` otherwise.

`isdigit()`

This returns `True` if every character of the string is a number, and `False` otherwise.

`strip()`

This returns a string in which the whitespace from the beginning and end of the original string is removed.

String Methods: Searching

`find(x)`

This returns the index of the location of `x` within the original string. It returns `-1` if `x` is not located.

`find(x, start)`

This returns the index of the location of `x` within the original string. It begins its search from the index `start`, and returns `-1` if `x` is not located.

String Methods: Substrings

`replace(x, y)`

This returns a string with every occurrence of `x` replaced by `y`.

`count(x)`

This counts the number of occurrences of `x` in the original string.

`startswith(x)`

This returns `True` if the original string begins with `x`, and `False` otherwise.

Escape Sequences

- Sometimes, you want to include special characters, such as the quotation mark, in your string.
- `Python` uses the backslash symbol, \ to accomplish this.

The newline character: `\n`

This character advances the cursor to the next line.

```
print("Hello\n\nworld")
```

The double quotation mark: `\"`

This inserts a double quotation mark into the string in the following manner:

```
print("She said, \"Hello\" to everyone.")
```

Escape Sequences

The backslash character: \\

This displays the backslash character.

```
print("C:\\Documents\\hello.py")
```

The tab character: \t

This inserts a tab character into the string. It is useful for producing tables.

```
print("Product\tWeight\tPrice")
print("kiwi\t0.15kg\t2.95")
```

Displaying Output with f-Strings

- f-strings are a new syntax in Python that allow you to easily include variables in strings.
- The string must be prefixed with the letter `f`, and the variables within the string must be enclosed with curly braces: `{}`

```
amount = 5
food = "pizza"
result = f"I had {amount} servings of {food}."
print(result)
```

Strings: End of Notes