Strings

Strings

In computer science, sequences of characters are referred to as *strings*. Strings can be any length and can include any character such as letters, numbers, symbols, and whitespace (spaces, tabs, new lines).

Indexing and Slicing Strings

Python strings can be indexed using the same notation as lists, since strings are lists of characters. A single character can be accessed with bracket notation (`[index]`), or a substring can be accessed using slicing (`[start:end]`).
Indexing with negative numbers counts from the end of the string.

```
str = 'yellow'
str[1]      # => 'e'
str[-1]     # => 'w'
str[4:6]    # => 'ow'
str[:4]     # => 'yell'
str[-3:]    # => 'low'
```

String Concatenation

To combine the content of two strings into a single string, Python provides the `+` operator. This process of joining strings is called concatenation.

```
x = 'One fish, '
y = 'two fish.'

z = x + y

print(z)
# Output: One fish, two fish.
```

Built-in Function `len()`

In Python, the built-in `len()` function can be used to determine the length of an object. It can be used to compute the length of strings, lists, sets, and other countable objects.

```python
length = len("Hello")
print(length)
# Output: 5

colors = ['red', 'yellow', 'green']
print(len(colors))
# Output: 3
```

IndexError

When indexing into a string in Python, if you try to access an index that doesn't exist, an `IndexError` is generated. For example, the following code would create an `IndexError`:

```python
fruit = "Berry"
indx = fruit[6]
```

Immutable strings

Strings are immutable in Python. This means that once a string has been defined, it can't be changed.
There are no mutating methods for strings. This is unlike data types like lists, which can be modified once they are created.

Escaping Characters

Backslashes (`\`) are used to escape characters in a Python string.
For instance, to print a string with quotation marks, the given code snippet can be used.

```python
txt = "She said \"Never let go\"."
print(txt) # She said "Never let go".
```

Iterate String

To iterate through a string in Python, "for...in" notation is used.

```python
str = "hello"
for c in str:
  print(c)

# h
# e
# l
```

```
# l
# o
```

The in Syntax

The `in` syntax is used to determine if a letter or a substring exists in a string. It returns `True` if a match is found, otherwise `False` is returned.

```python
game = "Popular Nintendo Game: Mario Kart"

print("l" in game) # Prints: True
print("x" in game) # Prints: False
```

String Method .lower()

The string method `.lower()` returns a string with all uppercase characters converted into lowercase.

```python
greeting = "Welcome To Chili's"

print(greeting.lower())
# Prints: welcome to chili's
```

String Method .upper()

The string method `.upper()` returns the string with all lowercase characters converted to uppercase.

```python
dinosaur = "T-Rex"

print(dinosaur.upper())
# Prints: T-REX
```

String Method .title()

The string method `.title()` returns the string in title case. With title case, the first character of each word is capitalized while the rest of the characters are lowercase.

```python
my_var = "dark knight"
print(my_var.title())

# Prints: Dark Knight
```

String Method `.split()`

The string method `.split()` splits a string into a list of items:

- If no argument is passed, the default behavior is to split on whitespace.

- If an argument is passed to the method, that value is used as the delimiter on which to split the string.

```python
text = "Silicon Valley"

print(text.split())
# Prints: ['Silicon', 'Valley']

print(text.split('i'))
# Prints: ['S', 'l', 'con Valley']
```

String Method `.join()`

The string method `.join()` concatenates a list of strings together to create a new string joined with the desired delimiter.
The `.join()` method is run on the delimiter and the array of strings to be concatenated together is passed in as an argument.

```python
x = "-".join(["Codecademy", "is", "awesome"])

print(x)
# Prints: Codecademy-is-awesome
```

String Method `.strip()`

The string method `.strip()` can be used to remove characters from the beginning and end of a string.
A string argument can be passed to the method, specifying the set of characters to be stripped.
With no arguments to the method, whitespace is removed.

```python
text1 = '   apples and oranges   '
text1.strip()        # => 'apples and oranges'

text2 = '...+...lemons and limes...-...'

# Here we strip just the "." characters
text2.strip('.')     # => '+...lemons and limes...-'

# Here we strip both "." and "+" characters
```

```
text2.strip('.+')    # => 'lemons and limes...-'

# Here we strip ".", "+", and "-" characters
text2.strip('.+-')   # => 'lemons and limes'
```

String replace

The `.replace()` method is used to replace the occurence of the first argument with the second argument within the string.

The first argument is the old substring to be replaced, and the second argument is the new substring that will replace every occurence of the first one within the string.

```
fruit = "Strawberry"
print(fruit.replace('r', 'R'))

# StRawbeRRy
```

Python string method `.find()`

The Python string method `.find()` returns the index of the first occurrence of the string passed as the argument. It returns `-1` if no occurrence is found.

```
mountain_name = "Mount Kilimanjaro"
print(mountain_name.find("o")) # Prints 1 in the console.
```

Python String .format()

The Python string method `.format()` replaces empty brace (`{}`) placeholders in the string with its arguments.

If keywords are specified within the placeholders, they are replaced with the corresponding named arguments to the method.

```
msg1 = 'Fred scored {} out of {} points.'
msg1.format(3, 10)
# => 'Fred scored 3 out of 10 points.'

msg2 = 'Fred {verb} a {adjective} {noun}.'
msg2.format(adjective='fluffy', verb='tickled', noun='hamster')
# => 'Fred tickled a fluffy hamster.'
```