String Processing

String Processing

String slicing and methods

String Review

String Review

```
>>> name = "Tyler" + " " + "Chang"
>>> print(name)
Tyler Chang
```

String Review

```
>>> name = "Tyler" + " " + "Chang"
>>> print(name)
Tyler Chang

>>> hi = "Hello world"
>>> hi == "Hello world"
True
>>> hi == "hello world"
False
>>> "Hello" < "hello"
True
```

String indexing

You can *index* Strings, just like a Tuple or List:

```
>>> vowels = "aeiou"
```

String indexing

You can *index* Strings, just like a Tuple or List:

```
>>> vowels = "aeiou"
```

```
"a e i o u"
 0 1 2 3 4
```

String indexing

You can *index* Strings, just like a Tuple or List:

```
>>> vowels = "aeiou"

             "a e i o u"
              0 1 2 3 4

>>> print(vowels[0])
a
```

String indexing

You can *index* Strings, just like a Tuple or List:

```
>>> vowels = "aeiou"

             "a e i o u"
              0 1 2 3 4

>>> print(vowels[0])
a

>>> print(vowels[-1])
u
```

String indexing

You can *index* Strings, just like a Tuple or List:

```
>>> vowels = "aeiou"

             "a e i o u"
              0 1 2 3 4

>>> print(vowels[0])
a

>>> print(vowels[-1])
u

>>> print(vowels[5])
IndexError:  string index out of range
```

Strings and Loops

Strings and Loops

```
>>> for letter in vowels:
...     print(letter)

a
e
i
o
u
```

Strings and Loops

```
>>> for letter in vowels:
...     print(letter)

a
e
i
o
u

for i in range(len(vowels)):
    print(vowels[i])

a
e
i
o
u
```

String Slicing

Can extract a substring by "slicing" a string, just like a list:

```
>>> name = "Tyler Chang"

>>> firstName = name[0:5]
>>> print(firstName)
Tyler

>>> print(name[:5])
Tyler

>>> print(name[-5:])
Chang
```

Mutability

Like Tuples, Strings are *immutable*.

Mutability

Like Tuples, Strings are *immutable*.

So the following fails:

Mutability

Like Tuples, Strings are *immutable*.

So the following fails:

```
>>> name = "Tyler"
```

Mutability

Like Tuples, Strings are *immutable*.

So the following fails:

```
>>> name = "Tyler"
>>> name[0] = "K"
```

Mutability

Like Tuples, Strings are *immutable*.

So the following fails:

```
>>> name = "Tyler"
>>> name[0] = "K"
TypeError:  "str" object does not support item
assignment
```

Mutability

Like Tuples, Strings are *immutable*.

So the following fails:

```
>>> name = "Tyler"
>>> name[0] = "K"
TypeError:  "str" object does not support item
assignment
```

This means that every "=" assignment for a String must create a copy:

Mutability

Like Tuples, Strings are *immutable*.

So the following fails:

```
>>> name = "Tyler"
>>> name[0] = "K"
TypeError:  "str" object does not support item
assignment
```

This means that every "=" assignment for a String must create a copy:

```
>>> nameCopy = name
```

Mutability

Like Tuples, Strings are *immutable*.

So the following fails:

```
>>> name = "Tyler"
>>> name[0] = "K"
TypeError:  "str" object does not support item
assignment
```

This means that every "=" assignment for a String must create a copy:

```
>>> nameCopy = name
>>> nameCopy += " Chang"
```

Mutability

Like Tuples, Strings are *immutable*.

So the following fails:

```
>>> name = "Tyler"
>>> name[0] = "K"
TypeError:  "str" object does not support item
assignment
```

This means that every "=" assignment for a String must create a copy:

```
>>> nameCopy = name
>>> nameCopy += " Chang"
>>> print(name)
```

Mutability

Like Tuples, Strings are *immutable*.

So the following fails:

```
>>> name = "Tyler"
>>> name[0] = "K"
TypeError:  "str" object does not support item
assignment
```

This means that every "=" assignment for a String must create a copy:

```
>>> nameCopy = name
>>> nameCopy += " Chang"
>>> print(name)
Tyler
```

Mutability

Like Tuples, Strings are *immutable*.

So the following fails:

```
>>> name = "Tyler"
>>> name[0] = "K"
TypeError:  "str" object does not support item
assignment
```

This means that every "=" assignment for a String must create a copy:

```
>>> nameCopy = name
>>> nameCopy += " Chang"
>>> print(name)
Tyler
>>> print(nameCopy)
```

Mutability

Like Tuples, Strings are *immutable*.

So the following fails:

```
>>> name = "Tyler"
>>> name[0] = "K"
TypeError:  "str" object does not support item
assignment
```

This means that every "=" assignment for a String must create a copy:

```
>>> nameCopy = name
>>> nameCopy += " Chang"
>>> print(name)
Tyler
>>> print(nameCopy)
Tyler Chang
```

Concept Check!

```
phrase = "Python is"
```

What is printed by each of the following commands?

1. `print(phrase + " cool!")`
2. `print(len(phrase))`
3. `print(phrase + phrase[:6])`
4. `print(phrase[:-3])`
5. `phrase[-1] = "z"`

Concept Check!

```
phrase = "Python is"
```

What is printed by each of the following commands?

1. `print(phrase + " cool!")` `Python is cool!`
2. `print(len(phrase))`
3. `print(phrase + phrase[:6])`
4. `print(phrase[:-3])`
5. `phrase[-1] = "z"`

Concept Check!

```
phrase = "Python is"
```

What is printed by each of the following commands?

1. `print(phrase + " cool!")` — `Python is cool!`
2. `print(len(phrase))` — `9`
3. `print(phrase + phrase[:6])`
4. `print(phrase[:-3])`
5. `phrase[-1] = "z"`

Concept Check!

```
phrase = "Python is"
```

What is printed by each of the following commands?

1. `print(phrase + " cool!")` — `Python is cool!`
2. `print(len(phrase))` — `9`
3. `print(phrase + phrase[:6])` — `Python is Python`
4. `print(phrase[:-3])`
5. `phrase[-1] = "z"`

Concept Check!

```
phrase = "Python is"
```

What is printed by each of the following commands?

1. `print(phrase + " cool!")`	`Python is cool!`
2. `print(len(phrase))`	`9`
3. `print(phrase + phrase[:6])`	`Python is Python`
4. `print(phrase[:-3])`	`Python`
5. `phrase[-1] = "z"`	

Concept Check!

```
phrase = "Python is"
```

What is printed by each of the following commands?

1. `print(phrase + " cool!")`	`Python is cool!`
2. `print(len(phrase))`	`9`
3. `print(phrase + phrase[:6])`	`Python is Python`
4. `print(phrase[:-3])`	`Python`
5. `phrase[-1] = "z"`	`TypeError`

Methods

```
name = " Tyler Chang "
```

Methods

```
name = " Tyler Chang "
```

- ```
  name.strip()
  "Tyler Chang"
  ```
  ```

Methods

```
name = " Tyler Chang "
```

- `name.strip()`
  `"Tyler Chang"`
- `name.split()`
  `["Tyler", "Chang"]`

Methods

```
name = " Tyler Chang "
```

- `name.strip()`
  `"Tyler Chang"`
- `name.split()`
  `["Tyler", "Chang"]`
- `name.lower()`
  `" tyler chang "`

Methods

```
name = " Tyler Chang "
```

- `name.strip()`
  `"Tyler Chang"`
- `name.split()`
  `["Tyler", "Chang"]`
- `name.lower()`
  `" tyler chang "`
- `name.upper()`
  `" TYLER CHANG "`

Methods

```
name = " Tyler Chang "
```

- `name.strip()`
  `"Tyler Chang"`
- `name.split()`
  `["Tyler", "Chang"]`
- `name.lower()`
  `" tyler chang "`
- `name.upper()`
  `" TYLER CHANG "`
- `" ".join(["Tyler", "Chang"])`
  `"Tyler Chang"`

Methods

```
name = " Tyler Chang "
```

- `name.strip()`
  `"Tyler Chang"`
- `name.split()`
  `["Tyler", "Chang"]`
- `name.lower()`
  `" tyler chang "`
- `name.upper()`
  `" TYLER CHANG "`
- `" ".join(["Tyler", "Chang"])`
  `"Tyler Chang"`

Read more:
`https://docs.python.org/3/library/stdtypes.html#string-methods`

string module

string module

There are additional string formatting functions and named constants in the `string` module, in the standard library.

string module

There are additional string formatting functions and named constants in the `string` module, in the standard library.

```
>>> import string
```

string module

There are additional string formatting functions and named constants in the `string` module, in the standard library.

```
>>> import string
```

Read more about them here:
`https://docs.python.org/3.8/library/string.html`