CS 100A Week 4

More strings, modulo, integer division, grids

Announcements

- Midterm: Tuesday November 1st from 6-7pm PST
- Midterm review next Thursday section
  - tons of practice problems
  - I'll bring snacks :)
- Please try and stay until the end of section (7:50pm)! If you have to leave early, **please** let me know beforehand.

String review

What's wrong with this code?

```
original = "Let's drive the car to the store."

original[18] = 't'
```

What's wrong with this code?

```
original = "Let's drive the car to the store."

original[18] = 't'
```

Strings in Python are **immutable.** That means that once you create a string, you can never change it.

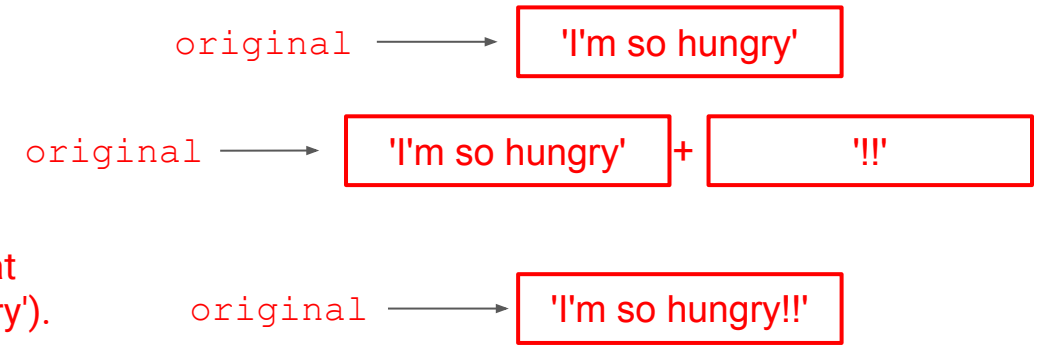
What's wrong with this code?

```
original = "Let's drive the car to the store."

original[18] = 't'
```

But what about string concatenation?

String concatenation and immutability

```
original = 'I'm so hungry'

original = original + '!!'
```



Here, we're not actually modifying the string that the variable `original` points to ('I'm so hungry').

We're actually creating an entirely new string, which is equal to the concatenation (sum) of the string that `original` points to and `'!!'`.

Then we're setting the variable `original` to point to that new string.

Fun Python Trick: -1 indexing

What is the last index of a string? `len(string)- 1`

In Python, you can use a shortcut index: `-1`

Index `-1` always refers to the last index of a string.

```
gourd = 'pumpkin'

last_char = gourd[-1]
```

???

Fun Python Trick: -1 indexing

What is the last index of a string? `len(string)- 1`

In Python, you can use a shortcut index: `-1`

Index `-1` always refers to the last index of a string.

```
gourd = 'pumpkin'

last_char = gourd[-1]
```
-> 'n'

String searching

Pattern:

```
substring in string ->
    True/False
```

String searching

```
def contains_cat(str):
    if 'cat' in str:
        return True
    else:
        return False
```

```
def does_not_contain_cat(str):
    if 'cat' not in str:
        return True
    else:
        return False
```

String searching

```
def contains_cat(str):
    if 'cat' in str:
        return True
    else:
        return False
```

```
def does_not_contain_cat(str):
    if 'cat' not in str:
        return True
    else:
        return False
```

Will the following statements evaluate to True or False?

1. `'mom' in 'I love my mom!'`

2. `'56' in '123456'`

3. `'aba' in 'abba'`

Will the following statements evaluate to True or False?

1. `'mom' in 'I love my mom!'` -> `True`

2. `'56' in '123456'` -> `True`

3. `'aba' in 'abba'` -> `False`

`find()` function

returns the first occurrence of a substring, or -1 if it doesn't exist

What does the following code evaluate to?

```
drink = 'coffee'
ff_position = drink.find('ff')
v_position = drink.find('v')
```

`find()` function

returns the first occurrence of a substring, or -1 if it doesn't exist

What does the following code evaluate to?

```
drink = 'coffee'
ff_position = drink.find('ff') # 2
v_position = drink.find('v') # -1
```

String slicing

Used to "slice" away a substring of a string

```
string[start:end]
```

String slicing

Used to "slice" away a substring of a string

```
string[start:end]
```

start index: index of first character to include in the substring slice

end index: index of first character to exclude from the substring slice

String slicing shortcuts

- `string[:end]` is equivalent to `string[0:end]`
  - the entire string up to and not including `end`

- `string[start:]` is equivalent to `string[start:len(string)]`
  - the entire string including and after `start`

What will the following string slices equal?

```
holiday = 'Halloween!'
```

1. `holiday[5:]`

2. `holiday[2:4]`

3. `holiday[:]`

What will the following string slices equal?

```
holiday = 'Halloween!'
```

1. `holiday[5:]` -> `'ween!'`

2. `holiday[2:4]` -> `'ll'`

3. `holiday[:]` -> `'Halloween!'`

Modulo (mod) operator

```
x % y
```

"x mod y"

Returns the **remainder** of x / y

What do the following statements evaluate to?

1. `4 % 2`
2. `1 % 3`
3. `11 % 2`

```
x % y
```

"x mod y"

Returns the **remainder** of x / y

What do the following statements evaluate to?

1. `4 % 2` 0
2. `1 % 3` 1
3. `11 % 2` 1

Grid Mini-Quiz

Given the grid

```
grid = Grid.build([ [1, 4, 89, 2],
                    [6, -23, 0, 345],
                    ['c', 90, -11, 'a']
                  ])
```

1	4	89	2
6	-23	0	345
'c'	90	-11	'a'

How would you get the following values from the grid using `grid.get(x, y)`?

1. 89
2. -11
3. 'a'
4. 90

Given the grid

```
grid = Grid.build([ [1, 4, 89, 2],
                    [6, -23, 0, 345],
                    ['c', 90, -11, 'a']
                  ])
```

	0	1	2	3
0	1	4	89	2
1	6	-23	0	345
2	'c'	90	-11	'a'

How would you get the following values from the grid using `grid.get(x, y)`?

1. 89 -> `grid.get(2, 0)`
2. -11
3. 'a'
4. 90

Given the grid

```
grid = Grid.build([ [1, 4, 89, 2],
                    [6, -23, 0, 345],
                    ['c', 90, -11, 'a']
                  ])
```

	0	1	2	3
0	1	4	89	2
1	6	-23	0	345
2	'c'	90	-11	'a'

How would you get the following values from the grid using `grid.get(x, y)`?

1. 89
2. -11 -> `grid.get(2, 2)`
3. 'a'
4. 90

Given the grid

```
grid = Grid.build([ [1, 4, 89, 2],
                    [6, -23, 0, 345],
                    ['c', 90, -11, 'a']
                  ])
```

	0	1	2	3
0	1	4	89	2
1	6	-23	0	345
2	'c'	90	-11	'a'

How would you get the following values from the grid using `grid.get(x, y)`?

1. 89
2. -11
3. 'a' -> `grid.get(3, 2)`
4. 90

Given the grid

```
grid = Grid.build([ [1, 4, 89, 2],
                    [6, -23, 0, 345],
                    ['c', 90, -11, 'a']
                  ])
```

	0	1	2	3
0	1	4	89	2
1	6	-23	0	345
2	'c'	90	-11	'a'

How would you get the following values from the grid using `grid.get(x, y)`?

1. 89
2. -11
3. 'a'
4. 90 -> `grid.get(1, 2)`

Practice Problems