Sets and Dictionaries

Camp Inc. Coding Track

Summer 2016

Warm Up: Spot the Bug(s)!

What is wrong with the snippet of code below?

```
pets = input("How many pets do you have? ")
if pets < 0:
    print("That's impossible!")
if pets = 0:
    print("Try pets sometime!")
else:
    print("Can I meet them?")
```

Warm Up: Spot the Bug(s)!

Corrected code snippet:

```
pets = int(input("How many pets do you have? "))
if pets < 0:
    print("That's impossible!")
elif pets == 0:
    print("Try pets sometime!")
else:
    print("Can I meet them?")
```

Review: Generating Ranges

The generator function `range` creates an iterable for looping over a sequence of numbers. This may be more convenient than using `while`.

The syntax is `range(`$start$`, `$stop$`, `$step$`)` if $start$ is not provided, it is assumed to be 0, and if $step$ is not provided, it is assumed to be 1.

Here is an example:

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

Warm Up: Trace the Loops!

```
for i in range(3):
    for j in range(2):
        print(i, j)
```

Warm Up: Trace the Loops!

```
for i in range(3):
    for j in range(2):
        print(i, j)
```

```
0 0
0 1
1 0
1 1
2 0
2 1
```

What Data Types Do We Know?

- Integers (eg. `19`)

What Data Types Do We Know?

- Integers (eg. `19`)
- Floating point numbers (eg. `19.5`)

What Data Types Do We Know?

- Integers (eg. `19`)
- Floating point numbers (eg. `19.5`)
- Booleans (`True` and `False`)

What Data Types Do We Know?

- Integers (eg. `19`)
- Floating point numbers (eg. `19.5`)
- Booleans (`True` and `False`)
- Strings (eg. `"Hello"`)

What Data Types Do We Know?

- Integers (eg. `19`)
- Floating point numbers (eg. `19.5`)
- Booleans (`True` and `False`)
- Strings (eg. `"Hello"`)
- Lists (eg. `[12, 13, ["Hello", "There"], "CI"]`)
  Ordered and **potentially non-unique** collections of data

Review: Lists

Lists are a data structure which allow us to store **ordered data**. We can specify list literals in Python using brackets.

```
my_list = [12, 13, ["Hello", "There"], "CI"]
```

Review: Lists

Lists are a data structure which allow us to store **ordered data**. We can specify list literals in Python using brackets.

```
my_list = [12, 13, ["Hello", "There"], "CI"]
```

Specific members of a list can be accessed by specifying the **zero-indexed** offset in brackets.

- `my_list[0]` is **?**

Review: Lists

Lists are a data structure which allow us to store **ordered data**. We can specify list literals in Python using brackets.

```
my_list = [12, 13, ["Hello", "There"], "CI"]
```

Specific members of a list can be accessed by specifying the **zero-indexed** offset in brackets.

- `my_list[0]` is `12`

Review: Lists

Lists are a data structure which allow us to store **ordered data**. We can specify list literals in Python using brackets.

```
my_list = [12, 13, ["Hello", "There"], "CI"]
```

Specific members of a list can be accessed by specifying the **zero-indexed** offset in brackets.

- `my_list[0]` is `12`
- `my_list[2][1]` is **?**

Review: Lists

Lists are a data structure which allow us to store **ordered data**. We can specify list literals in Python using brackets.

```
my_list = [12, 13, ["Hello", "There"], "CI"]
```

Specific members of a list can be accessed by specifying the **zero-indexed** offset in brackets.

- `my_list[0]` is `12`
- `my_list[2][1]` is `"There"`

Sets

Sets are like lists in that they are a container for multiple objects, however, they are **unordered** and cannot contain multiple copies of the same object. We can specify set literals in Python using squirly braces.

```
my_set = {'a', 12, 'ci', 'cheese'}
```

Sets

Sets are like lists in that they are a container for multiple objects, however, they are **unordered** and cannot contain multiple copies of the same object. We can specify set literals in Python using squirly braces.

```
my_set = {'a', 12, 'ci', 'cheese'}
```

We can check if an item exists in a set using the `in` operator.

- `'cheese' in my_set` will evaluate to **?**

Sets

Sets are like lists in that they are a container for multiple objects, however, they are **unordered** and cannot contain multiple copies of the same object. We can specify set literals in Python using squirly braces.

```
my_set = {'a', 12, 'ci', 'cheese'}
```

We can check if an item exists in a set using the `in` operator.

- `'cheese' in my_set` will evaluate to `True`

Sets

Sets are like lists in that they are a container for multiple objects, however, they are **unordered** and cannot contain multiple copies of the same object. We can specify set literals in Python using squirly braces.

```
my_set = {'a', 12, 'ci', 'cheese'}
```

We can check if an item exists in a set using the `in` operator.

- `'cheese' in my_set` will evaluate to `True`
- `42 in my_set` will evaluate to **?**

Sets

Sets are like lists in that they are a container for multiple objects, however, they are **unordered** and cannot contain multiple copies of the same object. We can specify set literals in Python using squirly braces.

```
my_set = {'a', 12, 'ci', 'cheese'}
```

We can check if an item exists in a set using the `in` operator.

- `'cheese' in my_set` will evaluate to `True`
- `42 in my_set` will evaluate to `False`

Example of Using a Set

`s.add(item)` will add `item` to the set `s` if `item` is not already in `s`.

Example of Using a Set

`s.add(item)` will add `item` to the set `s` if `item` is not already in `s`.

```
food = set()        # this creates an empty set
while True:
    line = input('Give a tasty food, blank to end: ')
    if line == '':
        break       # exits the while loop
    food.add(line)
print('You think', len(food), 'foods are tasty')
```

Sets vs. Lists

When should you choose a set over a list?

Sets vs. Lists

When should you choose a set over a list?

- You should choose a set if you have data in which **order does not matter**, and **cannot have multiple copies**.

Sets vs. Lists

When should you choose a set over a list?

- You should choose a set if you have data in which **order does not matter**, and **cannot have multiple copies**.
- You should choose a list if you have data in which **order matters**, and **multiple copies can exist**.

Sets vs. Lists

When should you choose a set over a list?

- You should choose a set if you have data in which **order does not matter**, and **cannot have multiple copies**.
- You should choose a list if you have data in which **order matters**, and **multiple copies can exist**.

There are certainly plenty of exceptions to the rules presented above, however, for the scope of this class, you should be fine following these rules.

Dictionaries

Dictionaries in Python are a key-value mapping. They are like a list, but elements are accessed using a key, rather than an index.

```
my_dict = {'mary': 'little lamb', 'joe': 'cow'}
```

Access is similar to a list, but the key replaces the offset.

Dictionaries

Dictionaries in Python are a key-value mapping. They are like a list, but elements are accessed using a key, rather than an index.

```
my_dict = {'mary': 'little lamb', 'joe': 'cow'}
```

Access is similar to a list, but the key replaces the offset.

- `my_dict['mary']` is **?**

Dictionaries

Dictionaries in Python are a key-value mapping. They are like a list, but elements are accessed using a key, rather than an index.

```
my_dict = {'mary': 'little lamb', 'joe': 'cow'}
```

Access is similar to a list, but the key replaces the offset.

- `my_dict['mary']` is `'little lamb'`

Dictionaries

Dictionaries in Python are a key-value mapping. They are like a list, but elements are accessed using a key, rather than an index.

```
my_dict = {'mary': 'little lamb', 'joe': 'cow'}
```

Access is similar to a list, but the key replaces the offset.

- `my_dict['mary']` is `'little lamb'`
- `my_dict['joe']` is **?**

Dictionaries

Dictionaries in Python are a key-value mapping. They are like a list, but elements are accessed using a key, rather than an index.

```
my_dict = {'mary': 'little lamb', 'joe': 'cow'}
```

Access is similar to a list, but the key replaces the offset.

- `my_dict['mary']` is `'little lamb'`
- `my_dict['joe']` is `'cow'`

Dictionaries

Dictionaries in Python are a key-value mapping. They are like a list, but elements are accessed using a key, rather than an index.

```
my_dict = {'mary': 'little lamb', 'joe': 'cow'}
```

Access is similar to a list, but the key replaces the offset.

- `my_dict['mary']` is `'little lamb'`
- `my_dict['joe']` is `'cow'`
- `my_dict['cow']` is **?**

Dictionaries

Dictionaries in Python are a key-value mapping. They are like a list, but elements are accessed using a key, rather than an index.

```
my_dict = {'mary': 'little lamb', 'joe': 'cow'}
```

Access is similar to a list, but the key replaces the offset.

- `my_dict['mary']` is `'little lamb'`
- `my_dict['joe']` is `'cow'`
- `my_dict['cow']` is Error
  - This results in a `KeyError` exception

Consider Dictionaries Like a Table

Having trouble with dictionaries? Think of them like a table, where the **key** is the column you look up an entry by, and the **value** is the column you are looking for.

Name (key)	**Phone No.** (value)
Alice	(123) 456-7890
Bill	(212) 555-1212
Jane	(444) 555-6666
Mary	(890) 123-4567
John	(791) 234-2255

Iterating over a Dictionary

Calling `.keys()` on a dictionary will give us an iterable of the keys. This allows us to loop like this:

```
d = {"cats": 10, "dogs": 15, "eggs": 20}
for key in d.keys():
    print("{}: {}".format(key, d[key]))
```

```
cats: 10
eggs: 20
dogs: 15
```

1

Iterating over a Dictionary

Calling `.keys()` on a dictionary will give us an iterable of the keys. This allows us to loop like this:

```
d = {"cats": 10, "dogs": 15, "eggs": 20}
for key in d.keys():
    print("{}: {}".format(key, d[key]))
```

```
cats: 10
eggs: 20
dogs: 15
```

What determined the order we got the keys? As the keys of a dictionary are like a set, they are unordered and unique, the iteration order was deterministically random[1].

[1] It was actually determined by the ordering from the string's hash function

Phonebook Program

The course website has an example program using a dictionary as a phone book. Download it, play with it, and maybe even remix your own.

Don't forget the documentation!

The *Data Structures* page in the official Python documentation has excellent information and examples on using lists, sets, and dictionaries.

These slides are nowhere near complete! Go forth and read the docs!