Python Reader

Introduction

Style

Interpreter

Variables

`print`

`input`

Math and expressions

Constants

random

Booleans

`if`

`while`

`for`

Functions

DocTests

Images

TextGrid

Lists

Strings

Dictionaries

Files

Nested Structures

Back to Code in Place

Dictionaries

What are Dictionaries?

Dictionaries associate a key with a value

- **Key** is a unique identifier
- **Value** is something we associate with that key

Here are some examples of dictionaries in the real world:

- Phonebook:
  - Keys: names
  - Values: phone numbers
- Dictionary
  - Keys: words
  - Values: word definitions
- US Government
  - Keys: Social Security number
  - Values: Information about an individual's employment

Dictionaries in Python

Creating dictionaries

- Dictionary start/end with braces: {}
- Key:Value pairs are separated by a colon
- Each pair is separated by a comma

Here is how to create a dictionary that stores the ages of the members of our teaching team: Chris, Brahm, and Mehran.

```
ages = {'Chris': 32, 'Brahm': 23, 'Mehran': 50}
```

Accessing elements of a dictionary:

- We can use keys to access their associated value

  ```
  ages['Chris'] # is 32
     ages['Mehran'] # is 50
  ```

- We can set values like regular variables

  ```
  ages['Mehran'] = 18 # sets Mehran's age to 18

  ages['Mehran'] += 3 # adds 3 to Mehran's age
  ```

- If we try to access a key that is not in the dictionary, we will get an error. For example, here is the console output if we try to find Brahm's age:

  ```
  >>> brahms_age = ages['Brahm']
  >>> brahms_age
  23
  ```

  However, we get an error if we try to access Santa Claus's age, which is not in the dictionary:

  ```
  >>> santas_age = ages['Santa Claus']
  KeyError: 'Santa Claus'
  ```

  KeyError means that the key we provided is not in the dictionary.

Python Reader

Introduction

Style

Interpreter

Variables

print

input

Math and expressions

Constants

random

Booleans

if

while

for

Functions

DocTests

Images

TextGrid

Lists

Strings

Dictionaries

Files

Nested Structures

Back to Code in Place

- We can use the keyword `in` to check if a key is in a dictionary, using a boolean expression to return true or false:

```
>>> 'Brahm' in ages
True
>>> 'Santa Claus' not in ages
True
```

Adding elements to a dictionary

- We can add and change pairs in a dictionary

```
phone = {} # starting with an empty dictionary
phone['Pat'] = '555-1212'
phone['Jenny'] = '867-5309'
phone['Pat'] = None
phone['Pat'] = '867-5309'
```

- Here's what the dictionary looks like after these names are added:

```
{'Pat': '867-5309', 'Jenny': '867-5309'}
```

A word about Keys/Values

- Keys must be immutable types (eg. int, float, string)
  - Keys cannot be changed in place
  - If you want to change a key, need to remove key/value pair from dictionary and then add key/value pair with new key.
- Values can be mutable or immutable types (eg. int, float, string, lists, dictionaries -- yes, you can have a dictionary of dictionaries!)
  - Values can be changed in place
- Dictionaries are mutable. This means changes made to a dictionary in a function persist after the function is done.

Let's take a further look at the significance of dictionaries being mutable. Here's an example, which takes our dictionary of ages and adds one to the age when a person has a birthday:

```
def have_birthday(dict, name):
    print("You're one year older, " + name + "!")
    dict[name] += 1

def main():
    ages = {'Chris': 32, 'Brahm': 23, 'Mehran': 50}
    print(ages)
    have_birthday(ages, 'Chris')
    print(ages)
    have_birthday(ages, 'Mehran')
    print(ages)
```

This is what the program prints out:

```
{'Chris': 32, 'Brahm': 23, 'Mehran': 50}
You're one year older, Chris!
{'Chris': 33, 'Brahm': 23, 'Mehran': 50}
You're one year older, Mehran!
{'Chris': 33, 'Brahm': 23, 'Mehran': 51}
```

Because dictionaries are mutable, changes made to the dictionary in the `have_birthday` function persist in the main function

Functions you can apply to dictionaries

We will now provide some useful functions you can apply to dictionaraies! We'll demonstrate them on our ages dictionary:

Python Reader

Introduction

Style

Interpreter

Variables

print

input

Math and expressions

Constants

random

Booleans

if

while

for

Functions

DocTests

Images

TextGrid

Lists

Strings

Dictionaries

Files

Nested Structures

Back to Code in Place

```
ages = {'Chris': 32, 'Brahm': 23, 'Mehran': 50}
```

dict.get(key)

Returns value associated with key in dictionary. Returns `None` if key doesn't exist.

```
>>> print(ages.get('Chris'))
32
>>> print(ages.get('Santa Claus'))
None
```

dict.get(key, default)

Returns value associated with key in dictionary. Returns default if key doesn't exist.

```
 >>> print(ages.get('Chris', 100))
32
>>> print(ages.get('Santa Claus', 100))
100
```

dict.keys()

Returns something similar to a range of the keys in dictionary. We can use this to loop over all the keys in a dictionary:

```
for key in ages.keys():
    print(str(key) + ", " + str(ages[key]))
```

We can also turn `keys()` into a list using the `list` function

```
>>> list(ages.keys())
   ['Chris', 'Brahm', 'Mehran']
```

We can also loop over a dictionary using for-each loop just using name of dictionary:

```
for key in ages:
    print(str(key) + ", " + str(ages[key]))
```

dict.values()

Returns something similar to a range of the values in dictionary. We can use this to loop over all the values in a dictionary:

```
for value in ages.values():
      print(value)
```

We can also turn `values()` into a list using the `list` function

```
>>> list(ages.values())
   [32, 23, 50]
```

dict.pop(key)

Removes key/value pair with the given key. Returns value from that key/value pair.

```
>>> ages
>>> {'Chris': 32, 'Brahm': 23, 'Mehran': 50}
>>> ages.pop('Mehran')
50
>>> ages
{'Chris': 32, 'Brahm': 23}
```

dict.clear()

Removes all key/value pairs in the dictionary.

```
>>> ages.clear()
>>> ages
{}
```

Python Reader

Introduction

Style

Interpreter

Variables

print

input

Math and expressions

Constants

random

Booleans

if

while

for

Functions

DocTests

Images

TextGrid

Lists

Strings

Dictionaries

Files

Nested Structures

Back to Code in Place

len(dict)

Returns number of key/value pairs in the dictionary

```
>>> ages
{'Chris': 32, 'Brahm': 23, 'Mehran': 50}
>>> len(ages)
3
```

del dict[key]

Removes key/value pair with the given key. It's similar to pop, but doesn't return anything.

```
>>> ages
{'Chris': 32, 'Brahm': 23, 'Mehran': 50}
>>> del ages['Mehran']
>>> ages
{'Chris': 32, 'Brahm': 23}
```