# Week 5 Lecture Recap

Covered in this week’s lectures:

1. [Strings & Text Processing](#_pzqgkr5ecn7j) (Friday May 14)
2. [Dictionaries](#_bn3epyq7rwsb) (Monday May 17)
3. [Files](https://codeinplace2021.github.io/pythonreader/en/filereading/) (Wednesday May 19)

## Strings & Text Processing

Covered in this lecture:

1. [Strings are immutable](#_b3xvq7knsa57)
2. [String Functions to know](#_pt22gesx1cl1)
3. [How to Process a String](#_4xiuwykhqq6t)
4. [mPedigree Puzzle](#_xq3zrfgyge19)

### Strings are immutable

Let's say we have the string:

s = "brahm"

Now, let's say that Brahm wants to change the string, so that the 'b' in his name is capitalized, like this: **"Brahm".** To do that, Brahm might want to write a line of code like this, the same way you would change an element of a list:

s[0] = "B"

But when he does this, Brahm runs into an error message:

TypeError: 'str' object does not support item assignment

This is a key difference between strings and lists: strings can't be mutated. The only way to change a string variable is to reassign it and make a new string. Immutability guarantees that string parameters won’t change. See [this link for an example](https://docs.google.com/presentation/d/1zh6LOlWNH6aBss-MnTo-ored48Hw1HupVmTsIC6v2U8/edit?usp=sharing).

### String Functions to know

|  |  |
| --- | --- |
| **.upper() .lower()** | **.upper()** returns a string with all the letters in upper case, and **.lower()** returns a string with all the letters in lower case.  s = "So long and thanks for all the fish"  >>> s.upper()  "SO LONG AND THANKS FOR ALL THE FISH"  >>> s.lower()  "so long and thanks for all the fish" |
| **.replace(char1, char2)** | **.replace(char1, char2)** replaces all instances of the first parameter with the second parameter  s = "So long and thanks for all the fish"  >>> s.replace("a", "e")  "So long end thenks for ell the fish"  >>> s.replace("s", "")  "So long and thank for all the fih" |
| **.find(char)** | **.find(char)** returns the first index of where the parameter (a character) occurs in the string. If the character is not in the string, the function returns -1.  s = "So long and thanks for all the fish"  >>> s.find("n")  5  >>> s.find("x")  -1 |
| **.strip()** | **.strip()** removes all the white space at the beginning and end of the string.  s = " So long and thanks for all the fish "  >>> s.strip()  "So long and thanks for all the fish" |
| **.split()** | **.split** splits the string into a list. If you don't pass any parameters, the string is split by spaces:  s = "So long and thanks for all the fish"  >>> s.split()  ["So", "long", "and", "thanks", "for", "all", "the", "fish"]  If you do pass a parameter, the string will be split by the parameter that you pass. So, if you pass in a comma as a parameter, the string will be separated by commas.  names = "Bruce,Diana,Victor,Barry,Clark,Arthur,Hal"  >>> names.split(",")  ["Bruce", "Diana", "Victor", "Barry", "Clark", "Arthur", "Hal"] |

More Useful String Functions:

>>> chant = "wakanda forever"

>>> stop\_balrog = "YOU SHALL NOT PASS"

>>> spaces = " "

>>> number = "42"

>>> chant.startswith("wak")

True

>>> stop\_balrog.startswith("you")

False

>>> chant.endswith("ver")

True

>>> chant.title()

"Wakanda Forever"

>>> chant.islower()

True

>>> spaces.isspace()

True

>>> number.isdigit()

True

Because strings are immutable, these functions don’t change the string and return a new string instead.

### How to Process a String

Processing a string involves transforming or inspecting the contents of the string

|  |  |
| --- | --- |
| If you need both the index (i) and the character (char), use a for i in range loop | If you need just the character (char), use a for-each loop |
| **for i in range(len(s)):**  **char = s[i]**  **# process char** | **for char in s:**  **# process char** |

We can process a string to reverse a string. Once we have the code to reverse a string, we can check to see if the string is a palindrome. A palindrome is a word that reads the same way from the front and the back. Here are some examples: kayak, A man, a plan, a canal - Panama! Here's a palindrome in Hindi: कडक

**def reverse\_string(s):**

**reverse = ""**

**for ch in s:**

**reverse = ch + reverse**

**return reverse**

**def is\_palindrome(str):**

**normalized = normalize(str)**

**rev = reverse\_string(normalized)**

**return normalized == rev**

**def normalize(str):**

**'''**

**This function removes all spaces and punctuation from a string**

**'''**

**normalized = ''**

**for ch in str:**

**if ch.isalpha():**

**normalized += ch.lower()**

**return normalized**

Click on [this link to see a visual representation](https://docs.google.com/presentation/d/10mrIw7dmqo9-eq0eGBDDzd03RSNyiT8x2aL6QW2sFHA/edit?usp=sharing) of how our reverse\_string function works.

### mPedigree Puzzle

A significant problem people with malaria face is with counterfeit medicine. In 2013, the UN estimated that 700,000 people die each year because they are given counterfeit drugs. Bright Simons, the founder of mPedigree, wanted to find a way to prevent malaria patients from receiving counterfeit medicine.

Here's how Bright Simon's solution works: Each box of medicine from the manufacturer has a label with a number on it. Someone can then take that number, and text it to mPedigree. If that number comes from a manufacturer, mPedigree will let them know. If that number comes from a counterfeiter who just put a random number on the box, mPedigree can tell the patient to not take the medicine.

You now have all the knowledge needed to implement Bright Simon's solution in Python. How can we write a function that generates these numbers for his solution?

N\_LABELS = 5000

def main():

for i in range(N\_LABELS):

rand\_part = pad(random.randint(0, 99999), 5)

unique\_part = pad(i, 4)

id = rand\_part + unique\_part

print(id)

def pad(num, length):

num\_string = str(num)

while len(num\_string) < length:

num\_string = "0" + num\_string

return num\_string

## Dictionaries

Covered in this lecture:

1. [What are Dictionaries?](#_7rx4bvzauncy)
2. [Dictionaries in Python](#_8zrlus55w9v)
3. [Functions you can apply to dictionaries](#_3fgeyjsdaxtu)

### What are Dictionaries?

* Dictionaries associate a ***key*** with a ***value***
  + Key is a *unique* identifier
  + Value is something we associate with that key
* Examples 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

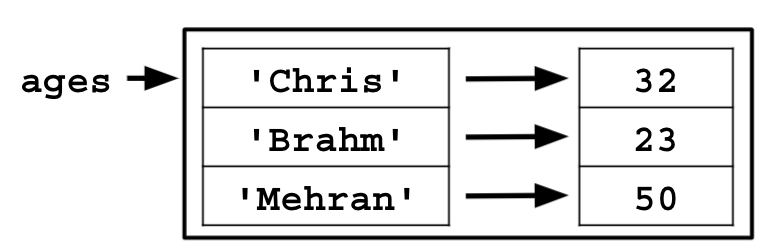
### Dictionaries in Python

* Creating dictionaries
  + Dictionary start/end with braces
  + Key:Value pairs separated by 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}**

The dictionary can be represented like this. Our keys on the left point to their associated values on the right.

****

Accessing elements of a dictionary:

* We can use keys to access their associated value

**ages['Chris']** is **32**

**ages['Mehran']** is **50**

* Can set values like regular variables:

**ages['Mehran'] = 18**

**ages['Mehran'] += 3**

* If we try to access a key that is not in the dictionary, we will get an error

**>>> brahms\_age = ages['Brahm']**

**>>> brahms\_age**

**23**

**>>> santas\_age = ages['Santa Claus']**

**KeyError: 'Santa Claus'**

* We can check if a key is in a dictionary:

**>>> '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'**

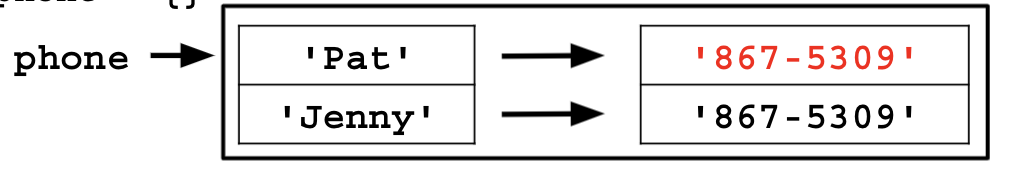
**phone['Jenny'] = '867-5309'**

**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
  + E.g., 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
  + E.g., int, float, string, lists, dictionaries
  + Values can be changed in place
* Dictionaries are mutable
  + Changes made to a dictionary in a function persist after the function is done.

For example, take a look at the following function, 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)**

**Terminal Output:**

|  |
| --- |
| **{'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

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

* Function: ***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**

* Function: ***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**

* Function: ***dict*.keys()**
  + Returns something similar to a range of the keys in dictionary
  + Can use that to loop over all keys in a dictionary:

**for key in ages.keys():**

**print(str(key) + ", " + str(ages[key]))**

* + Can turn **keys()** into a list, using the **list** function

**>>> list(ages.keys())**

**['Chris', 'Brahm', 'Mehran']**

* 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]))**

* Function: ***dict*.values()**
  + Returns something similar to a range of the values in dictionary
  + Can use that to loop over all values in a dictionary:

**for value in ages.values():**

**print(value)**

* + Can turn **values()** into a list, using the **list** function

**>>> list(ages.values())**

**[32, 23, 50]**

* Function: ***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}**

* Function: ***dict*.clear()**
  + Removes all key/value pairs in the dictionary.

**>>> ages.clear()**

**>>> ages**

**{}**

* Function: **len(*dict*)**
  + Returns number of key/value pairs in the dictionary

**>>> ages**

**{'Chris': 32, 'Brahm': 23, 'Mehran': 50}**

**>>> len(ages)**

**3**

* Function: **del *dict*[key]**
  + Removes key/value pairs in the dictionary.
  + Similar to **pop**, but doesn't return anything.

**>>> ages**

**{'Chris': 32, 'Brahm': 23, 'Mehran': 50}**

**>>> del ages['Mehran']**

**>>> ages**

**{'Chris': 32, 'Brahm': 23}**