ITEC 1150 Week 9 Chapter 6

TRUTHY / FALSEY, PROGRAMMING SHORTCUTS, & STRING OPERATIONS

Truthy / Falsey

THERE'S MORE TO TRUTH AND FALSENESS THAN JUST TRUE AND FALSE

What Does Truthy/Falsey Mean?

- Python (and many other languages) will evaluate lots of things besides `True` or `False` for conditions.
- For example, most strings and most numbers will be treated `True`, as will non-empty lists and dictionaries, which means those things are 'Truthy'.
- Empty lists, dictionaries, the empty string `''`, `0` and `0.0` will all be treated as `False`, which means those things are 'Falsey'.
- Let's look at that some examples, and then see it in action…

Truthy / Falsey Examples

```python
def truthy_falsey(thing):
    if thing:
        print(f'{thing} is True.')
    else:
        print(f'{thing} is False.')

truthy_falsey('') # empty string
truthy_falsey([]) # empty list
truthy_falsey({}) # empty dictionary
truthy_falsey(0)
truthy_falsey(0.0)
truthy_falsey('asdf')
truthy_falsey([1, 2, 3])
truthy_falsey({'key': 'value'})
truthy_falsey(1)
truthy_falsey(1.0)
```

```
 is False.
[] is False.
{} is False.
0 is False.
0.0 is False.
asdf is True.
[1, 2, 3] is True.
{'key': 'value'} is True.
1 is True.
1.0 is True.
```

Truthy / Falsey In Action

```python
while True:
    name = input("Enter a name, or just press enter to quit: ")
    if name:    # Anything other than an empty string is True!
        print(f'You entered {name}.')
    else:
        print("Goodbye!")
        break
```

- We could've done this:

```python
    if name != '':
```

- But using the power of Truthy is even easier, once you get the hang of it!
- **You don't need to use Truthy/Falsey in your programs, but you are welcome to do so!**

Output:

```
Enter a name, or just press
enter to quit: Joe

You entered Joe.

Enter a name, or just press
enter to quit:

Goodbye!
```

Programming Shortcuts

A REFRESHER ON HOW EXPRESSIONS ARE EVALUATED

Expression Refresher

- An expression is one or more values combined with an operator that evaluate down to a single value:

```
2 * math.pi
3 / 4
length = 5
width = 4
```

- The lines above are all single expressions. The line below contains *two* expressions—can you see both of them?

```
area = length * width
```

Expression Evaluation

- In the line

```
area = length * width
```

- the first expression is `length * width`, and the second expression is the assignment to the `area` variable.
- We already know that `length * width` is evaluated before the assignment happens.
- What if we had a function that needed to return the area?

Skipping Variable Assignment

- If we have a function that returns the result of an expression, most of us have been writing it like so:

```python
def calc_area(length, width):
    area = length * width
    return area
```

- Because `length * width` is evaluated first, we can skip the variable assignment, and just return the result of the expression directly:

```python
def calc_area(length, width):
    return length * width
```

- The second version is shorter and simpler and is the way experienced programmers would write the function.
- **You don't need to do this in your programs, but you are welcome to do so, and you *do* need to understand what's happening here!**

String Operations

MORE DETAILS ON ONE OF THE MOST COMMON DATA TYPES

We Know All About Strings, Don't We?

- We can multiply strings

```python
college = 'Minneapolis '
print(college * 3) # string multiplication
```

- And we have 3 options for concatenation

```python
result = 100
print('Your score: ' + str(result)) # book method
print('Your score:', result) # shorter – adds space
print(f'Your score: {result}') # .format() method
```

- Substituting variables with formatting codes is a snap

```python
speed = 4.2
goal = 1000
print(f'At Run 4 Kids, you averaged {speed:.1f} mph and raised ${goal:,.2f}.')
```

- We Even Love Tables

```python
print('Run 4 Kids')
print(f'{"MPH":<10}{"Goal":>10}')
print(f'{speed:<10.1f}${goal:>9,.2f}.')
```

```
Run 4 Kids
MPH             Goal
4.2       $ 1,000.00.
```

We Know About Escape Characters

" \' \" \t \n \\"

Run this code to demonstrate:

1. `print('This is Alice\'s cat')`
2. `print('\nAlice says, "What\'s her coat type?"')`
3. `print('\nShe\'s called a \t\t tabby cat.')`
4. `print('\nWhat does Alice have? \nA tabby cat!')`
5. `print('\nAlice\'s pet is of the feline\\cat family.')`

Multiline Strings May Be Easier At Times

- Instead of using a lot of escape characters, you can use multiline strings, starting and ending with 3 quotes.
- Run this code:

```python
print('''
Dear Alice,
    Just wanted to say, "Hello" and let you know how it's going with cat sitting. Your cat has been naughty/nice. She caught a mouse!
Happy travels,
Mo
''')
```

- Note that there's no real difference between a triple-quoted string and a triple-quoted comment—a multi-line string is a comment just because it isn't printed or assigned to a variable!

More on Comments

- We have been using # to add short comments anywhere in our programs

```
# This section calculates the total bill
```

- A multi-line comment in '''triple quotes''' can likewise occur in any location.

```
'''The next section of the program publishes the menu options:
View, Add, Edit, Delete'''
```

PyCharm suggests using """triple-double quotes""" when a multi-line comment is the special case called the docstring. This type of comment contains key information about a module, class or function.

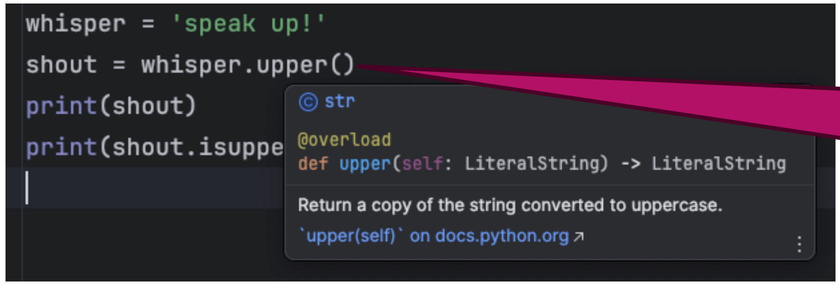
```
"""Author: Mo Lee Date: 1/1/2049
The content and format of docstrings is elaborated in pep 8 and each development shop.
We could have a whole lesson on this topic! """
```

PyCharm Tip

Most Integrated Development Environments (IDEs) do more than format and run code.

They can also provide help by hovering over methods or function names.

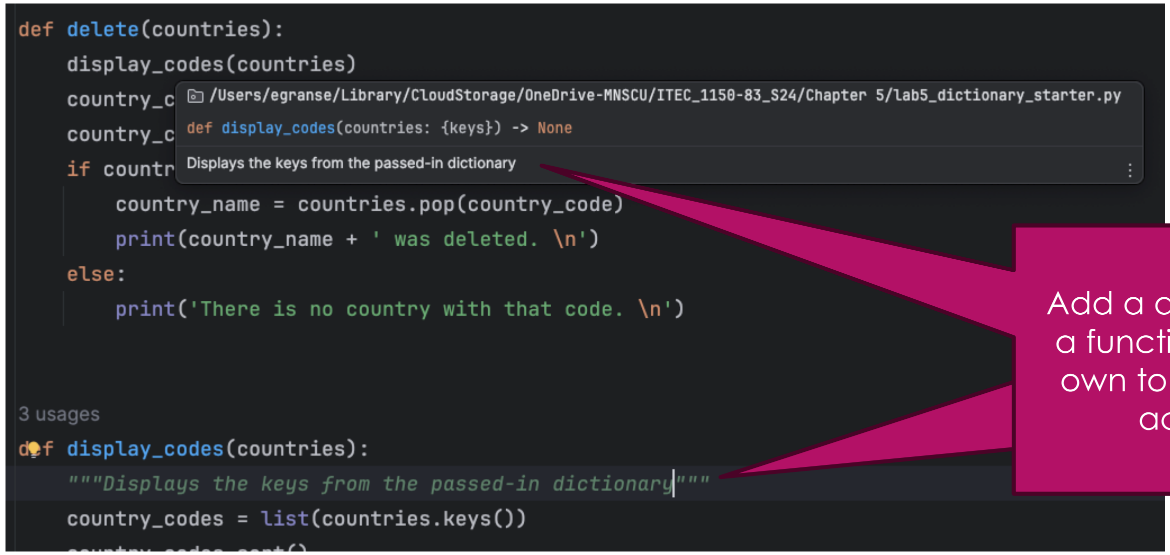
The amount of help usually depends on the amount of documentation added by the function or method developer.



```
whisper = 'speak up!'
shout = whisper.upper()
print(shout)
print(shout.isuppe
```

```
str
@overload
def upper(self: LiteralString) -> LiteralString
Return a copy of the string converted to uppercase.
`upper(self)` on docs.python.org ↗
```

Hovering the mouse over .upper() causes documentation to appear.



```
def delete(countries):
    display_codes(countries)
    country_c
    country_c
    if countr
        country_name = countries.pop(country_code)
        print(country_name + ' was deleted. \n')
    else:
        print('There is no country with that code. \n')


3 usages
def display_codes(countries):
    """Displays the keys from the passed-in dictionary"""
    country_codes = list(countries.keys())
```

```
/Users/egranse/Library/CloudStorage/OneDrive-MNSCU/ITEC_1150-83_S24/Chapter 5/lab5_dictionary_starter.py
def display_codes(countries: {keys}) -> None
Displays the keys from the passed-in dictionary
```

Add a docstring to a function of your own to see this in action.

A String Is Like A List Of Characters

A	n	d	r	o	i	d
0	1	2	3	4	5	6

- Since humans use strings to communicate, it's helpful to be able to process strings in more sophisticated ways
- Like a list, a string is <u>iterable</u>. Each character has an index, as does a list item.

- Use a loop to operate on each character.
- Let's review:

```python
phone = 'Android'
print(phone[5]) # prints the letter i
print(len(phone)) # prints 7
for i in 'Android':
    print(i.upper(), end = " ") # try with and without end = " "
```

⚠️ len(str) != Index of Last Character!

- Try running this code. What happens?

```python
phone = 'Android'
length = len(phone) # returns 7
last_letter = phone[length] # causes index error
print(last_letter)
```

- If you want to access the last character in a string, its index is the <u>string length – 1</u>

```python
# shorthand to access the last letter in a string variable
print(phone[len(phone)-1])
```

The `in` Operator: Methods For Validation

- Check if a string is part of another string – one way to validate input
- Do you have an "e" in your name? Run this code, and test names with and without an "e"

```
name = input('Enter a name: ')
if 'e' in name:
    print('This name has the letter e.')
else:
    print('There\'s no letter e in this name.')
```

- Instead of hard-coding a search for the letter "e," you could let the user decide what to search for. How would you revise the program to do that?

The in Operator (cont.)

- Is a particular email address likely to be a school email?

```python
email = input('Enter your email address: ')
if '.edu' in email:
    print('This is a school email.')
```

Adapt the program in your exercise file:

- Add an appropriate else message, in case the email doesn't contain the search term.
- Add an input statement, to let the user determine the search term for various top-level domains (.edu, .com, .gov, etc.).
- Test with several possibilities.

More String Methods - Uppercase, Lowercase

- Try the code below using different methods:
- `upper(), isupper()`

```
whisper = 'speak up!'
shout = whisper.upper()
print(shout)
print(shout.isupper())
```

- Try the code below using different methods:
- `lower(), islower()`

```
shout = 'TONE IT DOWN!'
whisper = shout.lower()
print(whisper)
print(whisper.islower())
```

Logical String Methods

- Logical methods are what we call methods that return Boolean types (True or False).
- In addition to `isupper()` and `islower()`, these other logical methods exist on strings:

• `isalpha()`	alphabetic characters only
• `isalnum()`	letters and numbers only (alpha-numeric)
• `isdecimal()`, `isnumeric()`, `isdigit()`	numbers 0-9 only (with a few exceptions)
• `isspace()`	spaces, tabs, and newlines
• `istitle()`	first letter of each word is upper case, followed by lower case

→ `issentence()` doesn't exist! How could you check if the first word of a multi-word string is capitalized?

→ How about checking if the last character is a period or other punctuation mark?

Practice With These Strings & Methods

```python
string1 = 'abcdef'
string2 = 'abc123'
string3 = '123456' # Strings are NOT numbers unless converted
string4 = '123.456' # Even if they look like numbers!
string5 = '\t \n'
string6 = 'Hello World'
```

```python
print('Is string1 alpha? ', string1.isalpha())
print('Is string2 alpha? ', string2.isalpha())
print('Is string2 alnum? ', string2.isalnum())
print('Is string1 alnum? ', string1.isalnum())
print('Is string3 a number? ', string3.isnumeric())
print('Is string4 a number? ', string4.isnumeric())
print('Is string4 convertible? ', float(string4))
print('Is string5 spacey? ', string5.isspace())
print('Is string6 title? ', string6.istitle()
```

```
Is string1 alpha?  True
Is string2 alpha?  False
Is string2 alnum?  True
Is string1 alnum?  True
Is string3 a number?  True
Is string4 a number?  False # Why not?
Is string4 convertible?  123.456
Is string5 spacey?  True
Is string6 title?  True
```

Changing Values Within Strings

- The `replace()` method:

```python
spelling_mistakes = 'I am going too school too learn programming.'
correct = spelling_mistakes.replace('too', 'to')
print(correct)
sentence = 'My ca=t pres==ses= the equals= key= when I ty=pe.'
new_sentence = sentence.replace('=', '')
print(new_sentence)
```

- Common errors and good things to fix routinely in user input are things such as:
  - Replace 2 spaces with 1
  - Deleting leading and trailing spaces
  - Fixing common typing mistakes ('adn' → 'and')

Removing Whitespace…

- You can use `.strip()`, `.rstrip()` and `.lstrip()` methods to remove spaces.
- To see the difference, try this code:

```
stuff = input('Type in a sentence with 3 spaces at the beginning and at the end')
print('1 ' + stuff + '!')
print('2 ' + stuff.strip() + '!')
print('3 ' + stuff.rstrip() + '!')
print('4 ' + stuff.lstrip() + '!')
```

Splitting Strings Into Lists

- Example 1 – run this code

```
sentence = 'This is a sentence.'
word_list = sentence.split()
print(word_list)
```

The difference between these examples is key to the lab problems

----

- Example 2 - Copy the block above and adapt as follows:
  - Include two sentences in the string, each ending in a period.
  - Change `.split()` method to `.split('.')`
  - Study the output. We can remove the extra list item with `.pop()`, but how about that pesky character at the beginning of the 2nd list item?

Splitting Strings...More Options

- Example 2 – adaptation gets rid of empty list item AND extra characters

```
sentences = 'This is a sentence. This is too.'
sentence_list1 = sentences.split('.') # has empty item
sentence_list1.pop() # gets rid of empty item
print(sentence_list1) # good, but 2nd item has leading space
sentence_list2 = []
for index in range(len(sentence_list1)):
    sentence_list2.append(sentence_list1[index].strip())
print(sentence_list2) # all good, but a pain!
```

Output

```
['This is a
sentence', ' This
is too']

['This is a
sentence', 'This is
too']
```

Splitting Strings…Yet More Options

- Example 3 – A bit shorter example that does the same thing as Example 2:

```python
sentences = 'This is a sentence. This is too.'
sentence_list2 = []
for sentence in sentences.split('.'):
    if sentence: # Truthy! If an item is an empty string, it's skipped!
        sentence_list2.append(sentence.strip())
print(sentence_list2)
```

Output

```
['This is a
sentence', 'This is
too']
```

Working With The Lists

```python
sentence = 'This is a word list.'
word_list = sentence.split()
print(word_list)
print('Here is your word list fixed & printed nicely')
new_word_list = []
for index in range(len(word_list)):
    if index < len(word_list) - 1:
        new_word_list.append(word_list[index].lower().strip())
        print(f'Word {index +1}: {new_word_list[index]}')
    else:
        new_word_list.append(word_list[index].lower().strip().replace(".", ''))
        print(f'Word {index + 1}: {new_word_list[index]}')
print(new_word_list)
```

Output:

```
Here is your word list fixed &
printed nicely

Word 1: this

Word 2: is

Word 3: a

Word 4: word

Word 5: list

['this', 'is', 'a', 'word',
'list']
```