Discussion Section # 2 : Iterables and Conditionals

Mitchell Valdes

Introduction

- **files needed** = None

Iterables

Lists

- Lists are a collection of items that are ordered and changeable.
- Lists are written with square brackets.

```
#| echo: true
# Example of a list
list1 = ["apple", "banana", "cherry", 1 , 2, True, ["1", "apple"]]

# We can access elements of a list using the index (starting at 0)
print(list1[2])

# Note that the last element of the list is a list itself
print(list1[-1])

# We can add elements to the list using the append method
list1.append("orange")
print(list1[-1])
```

Tuples

- Tuples are a collection of items that are ordered and unchangeable.
- Tuples are written with round brackets.

```
# Example of a tuple
tuple1 = ("apple", "banana", "cherry", 1 , 2, True, ["1", "apple"])

# We can access elements of a tuple using the index (starting at 0)
print(tuple1[2])
```

```
cherry
```

. . .

- If we try to change an element of a tuple, we will get an error.

```
tuple1[2] = "orange"
```

```
'tuple' object does not support item assignment
```

Dictionary

- Dictionaries are a collection of items that are unordered, changeable and indexed.
- Dictionaries are written with curly brackets.

```
# Example of a dictionary
car1 = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

# We can access elements of a dictionary using the key
print(car1["brand"])
```

```
Ford
```

Dictionaries are mutable

```
# We can change the value of a key
car1["year"] = 2020

# Or add a new key
car1["color"] = "red"

print(car1)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020, 'color': 'red'}
```

Retrieving Keys and Values

- We can retrieve the keys and values of a dictionary using the `keys()` and `values()` methods.

```
# Retrieve the keys of a dictionary
print(car1.keys())
```

```
dict_keys(['brand', 'model', 'year', 'color'])
```

```
# Retrieve the values of a dictionary
print(car1.values())
```

```
dict_values(['Ford', 'Mustang', 2020, 'red'])
```

Conditionals

`if` statements

- `if` statements are used to check if a condition is true or false.
- `if` statements are written with the `if` keyword.

```
# Example of an if statement
VOTING_AGE = 18
```

```
age = 17

if age >= VOTING_AGE:
    print("You are old enough to vote!")
```

`else` statements

- `else` statements are used to execute code if the condition is false.
- `else` statements are written with the `else` keyword.

```
# Complete the code below to print
# "You are not old enough to vote!"
# if the age is less than VOTING_AGE

if age >= VOTING_AGE:
    print("You are old enough to vote!")
```

More on conditionals: `elif`

We covered the basics of comparisons and conditional statements in class. Let's dig a bit deeper.

We have seen code of the form

```
if statement:
    'stuff to do if statement is true'
else:
    'stuff to do if statement is false'
```

More on conditionals: `elif`

What if our statement may have more than just two answers? We can use the `elif` (as in *else if*) construction.

```
status = 'fresh'

if status == 'fresh':
    class_rank = 0
```

```
elif status == 'soph':
    class_rank = 1
elif status == 'jun':
    class_rank = 2
else:
    class_rank = 3

print(class_rank)
```

```
0
```

Try changing `status` to 'jun'. What happens?

Now try changing `status` to 'super senior'. What happens?

The final `else` is the default value. If none of the preceding statements evaluate to `True` then the `else` clause is executed.

- You can have as many `elif`s as you like, but if you find yourself writing lots of `elif`s you should spend a few minutes considering other ways to implement your algorithm.
- You can leave off the final `else`, which would mean there is no default value. You should be careful, though, because `class_rank` is not defined for seniors. If you try to use it later, you will get an error.
- Once an `if` or `elif` is found to be true, the rest of the code block is skipped. In the example above, if `status=='fresh'` then none of the `elif`s are checked and neither is the `else`. This is the advantage of using `elif` statements rather than many if-else statements. Using multiple `if` statements would lead to all four conditions being checked separately, even if we are satisfied with the first check.

More on conditionals: `elif`

Are these equivalent?

case 1

```
like = 0
animal = "puppy"
```

```
if animal == "puppy":
    like = 1
```

case 2

```
animal = "puppy"

if animal == "puppy":
    like = 1
else:
    like = 0
```

If they're equivalent, what are the pros/cons of each style?

Looking in collections: `in`

`in` gives us an easy way to see if a value is contained in a collection.

```
names = ['kim', 'mitchell', 'bucky', 'barry', 'abe', 'hector', "satyen"]

name_to_check = 'BUCKY'

# Use the `in` keyword to check if a name is in the list
# Note that capitalization matters (how can we fix this?)
```

Looking in collections: `in`

```
jobs = {'kim':'prof',
        'mitchell':'TA',
        'bucky':'mascot',
        'barry':'football guy', 'abe':'president',
        'hector':'biochemist',
        'satyen':'TA'}

employee = 'kim'

# Complete the code below to check if Kim works at UW
# and print the his job
```

```
# Replace condition with the correct condition
condition = True

if condition:
    pass
else:
    print("Kim does not work at UW")

# Note pass is a keyword that does nothing.
```

Boolean logic: `not`

We have seen how to evaluate statements that return bool types. Things like

```
x = 10
y = 3
b = (x == y)
```

The `not` operator turns `False` into `True` and `True` into `False`. In mathematical terms, this is called *negation.*

```
print(not True)
```

```
False
```

```
print(not False)
```

```
True
```

Practice: Flow control

Question 1

Write code that controls a car at a stoplight. The variable `light` can take values 'green', 'yellow', or 'red'. Check the value of `light` and print out 'go', 'prepare to stop' and 'stop' as appropriate.

```
# Question 1

light = 'red'
```

Question 2

Without running any code, evaluate whether these statements are True or False.

1. `3 > 2 > -1`
2. `(3 > 2) > -1`
3. `'bill' != 'Bill'`
4. `not ('bill' != 'Bill')`
5. `(3<2) or not (2>3)`

Question 3

Below is the text of the Gettysburg Address. Write code that checks whether each phrase in

```
phrases = ['last full measure', 'of the people, by the people', 'four score and seven']
```

is in the Address. If the phrase is in the Address, then print out: Abe said '{phrase}.' If not, then print out: Abe did not say '{phrase}'.

Where you replace {phrase} with the phrase you are checking. Notice that the phrase is printed out with single quotation marks around it.

[Hint: A string is a collection, like a list or a dict.]

```
getty = """
    Four score and seven years ago our fathers brought forth on this continent, a new nati
    conceived in Liberty, and dedicated to the proposition that all men are created equal.
    Now we are engaged in a great civil war, testing whether that nation, or any nation so
    and so dedicated, can long endure. We are met on a great battle-field of that war. We
    dedicate a portion of that field, as a final resting place for those who here gave the
    that that nation might live. It is altogether fitting and proper that we should do thi
    But, in a larger sense, we can not dedicate we can not consecrate we can not hallow-th
    The brave men, living and dead, who struggled here, have consecrated it, far above our
    to add or detract. The world will little note, nor long remember what we say here, but
    forget what they did here. It is for us the living, rather, to be dedicated here to th
    which they who fought here have thus far so nobly advanced. It is rather for us to be
```

```
    great task remaining before us that from these honored dead we take increased devotion
    which they gave the last full measure of devotion that we here highly resolve that
    these dead shall not have died in vain that this nation, under God, shall have a new b
    freedom and that government of the people, by the people, for the people, shall not pe
    """
```

```
# Question 3
#| echo: true
phrases = ['last full measure', 'of the people, by the people', 'four score and seven']
```

Question 4

Here is a tweet:

```
tweet = """
#SalvadorDali The art of Salvador Dali never ceases to amaze me.
His surrealism is truly one of a kind.
If you're ever in NYC, make sure to check out the Dali exhibit at
@museumofmodernart!    #art #surrealism #SalvadorDali
"""
```

- We want to identify the hashtags and mentions in the tweet. Hashtags are words that start with a `#` and mentions are words that start with a `@`.
- We can use the `split()` method to split the tweet into a list of words.

```
# Split the tweet into a list of words
words = tweet.split()
print(words)
```

```
['#SalvadorDali', 'The', 'art', 'of', 'Salvador', 'Dali', 'never', 'ceases', 'to', 'amaze',
```

Complete the code below to print out the hashtags and mentions in the tweet.

If the workd is a hashtag, print out: **Hashtag: {word}**

If the word is a mention, print out: **Mention: {word}**

```
# Iterate through the words in the tweet
# We are using a for loop, we will cover this in the next discussion section
for word in words:
    pass
    # code goes here
```

Fin