mutability (class slides)

CSc 110 - Mutability

Lists vs strings

- Lists are **mutable**
- Strings are **not**
- In addition to retrieving an item from a `list`, we can remove or change it (which is **not** something you can do with strings)

```
songs = ["Lavender Haze", "Calm Down", "As It Was", "About Damn Time"]
songs
```

```
['Lavender Haze', 'Calm Down', 'As It Was', 'About Damn Time']
```

```
songs[0] = "Flowers"
songs
```

```
['Flowers', 'Calm Down', 'As It Was', 'About Damn Time']
```

Lists are mutable

(strings are not)

Because lists are mutable:

- once a list is changed **inside** a function, that **change persists** when the function has finished running
- if your function changes a list, you don't strictly need to return that list – we will be returning because that's what you need to be doing in the future

Write a function

1. Its name is `make_all_even`
2. It takes one argument, a list of `integers`
3. It iterates over the list, changing odd numbers to even number (even up)

```
test_integers = [1, 2, 3, 4]
assert make_all_even(test_integers) == test_integers
assert test_integers == [2, 2, 4, 4]
```

Write a function – solution

```
def make_all_even(integers):
  index = 0
  while index < len(integers):
    if integers[index] % 2 == 1:
      integers[index] += 1
    index += 1
  return integers

def main():
  test_integers = [1, 2, 3, 4]
  assert make_all_even(test_integers) == test_integers
  assert test_integers == [2, 2, 4, 4]
  print(test_integers) # we print the list we created before function call

main()
```

```
[2, 2, 4, 4]
```

Let's visualize this on Python Tutor

Object References

- A variable doesn't store values, it stores a reference to an object that lives in your computer memory (RAM)
- The same variable name can be pointed to a different object
- A variable name can point to an object that is already referenced by another variable

Examples

Strings are **not** mutable

```python
title = "Dr."
last_name = "Brown"
print(title + " " + last_name)

title = "Ms."
print(title + " " + last_name)

name = last_name
last_name = "Silva" # last_name points to a different object
print(title + " " + name)
```

```
Dr. Brown
Ms. Brown
Ms. Brown
```

Visualize these examples in Python Tutor

Examples

Lists are **mutable**

```python
names = ["Dr.", "Brown"]
print(names[0] + " " + names[1])

names[0] = "Ms."
print(names[0] + " " + names[1])

names_copy = names
names[1] = "Silva"
print(names_copy[0] + " " + names_copy[1])
```

```
Dr. Brown
Ms. Brown
Ms. Silva
```

Visualize these examples in Python Tutor

Another example

```
numbers = [50, 30, 80]
same_numbers = numbers
same_numbers[1] = 40
numbers = "look, numbers!"
same_numbers = numbers
```

Visualize these examples in Python Tutor

Object references

- A variable does not actually hold the value of the object within it
- Instead, it's a **reference** to the object
    - The object is "sitting" somewhere in your computer's memory (RAM)

Object references

- When you assign a value to a new variable, one of two things could happen
    - If you assign it to an existing object, the variable references that object
    - If you assign it to a new object, the object is created, placed in memory, and then the variable references it

Garbage Collection

- When there's no variable pointing to an object anymore, that object is removed from memory by Python's Garbage Collector (so you don't have to worry about memory leaks)

Summary

When working with lists, once they are changed in a function, the changes happen to the object in memory

Changes to lists persist once the function has finished running

.pop() list method

We will be using a few built-in list `methods` (not all of them, some you will implement yourself from scratch)

Here's how `.pop()` works:

```
songs = ["Lavender Haze", "Calm Down", "As It Was", "About Damn Time"]
songs
```

```
['Lavender Haze', 'Calm Down', 'As It Was', 'About Damn Time']
```

```
songs.pop(0)
songs
```

```
['Calm Down', 'As It Was', 'About Damn Time']
```

Write two functions

1. Names are `remove_vowels_list` and `remove_vowels_string`
2. The first function takes a list of `characters` as argument, the second takes a single `string`
3. The first function removes (use `.pop(index)`) all vowels from the `characters` list, the second creates a `new_string` with only characters that are not vowels
4. The first function returns the original argument list, the second function returns the `new_string`

```
assert remove_vowels_list(["b", "a", "n", "a", "n", "a"]) == ["b", "n", "n"]
assert remove_vowels_string("banana") == "bnn"
```

Write two functions – solution

```
def remove_vowels_list(characters):
  index = 0
  while index < len(characters):
    if characters[index] in "aeiou":
      characters.pop(index)
    else:
      index += 1 # go to next index only if no item has been removed
  return characters

def remove_vowels_string(string):
  new_string = ""
  index = 0
  while index < len(string):
    if string[index] not in "aeiou":
      new_string += string[index]
    index += 1
  return new_string

def main():
  test_characters = ["b", "a", "n", "a", "n", "a"]
  test_string = "banana"

  assert remove_vowels_list(test_characters) == test_characters
  assert test_characters == ["b", "n", "n"]
  assert remove_vowels_string("banana") == "bnn"

  print(test_characters)
  print(test_string)

main()
```

```
['b', 'n', 'n']
banana
```

Quiz 06

current time

You have 10 minutes to complete the quiz

- No need for comments, no need for a `main()`, no test cases
- Just write your function and what's inside the function
- USE A WHILE LOOP

Built-in functions you can use: `round()`, `input()`, `float()`, `str()`, `int()`, `len()` — you don't have to use all of these

List methods

We will be using the following list methods in this class:

- `.append()` adds an element at the end of the list: `list.append(value)`
- `.insert()` adds an element at the provided index: `list.insert(index, value)`
- `.pop()` removes a specific element at the provided index: `list.pop(index)`
- `.remove()` removes the first element with the provided value: `list.remove(value)`

`.append()` list method

```python
songs = ["Lavender Haze", "Calm Down", "As It Was", "About Damn Time"]
songs
```

```
['Lavender Haze', 'Calm Down', 'As It Was', 'About Damn Time']
```

```python
songs.append("Flowers")
songs
```

```
['Lavender Haze', 'Calm Down', 'As It Was', 'About Damn Time', 'Flowers']
```

Write a function

1. Its name is `indices_of_vowels`
2. It takes a single `string` as its parameter.
3. It returns a list of integers that represent the indices of the vowels in the original list

Test cases:

```python
assert indices_of_vowels("hello") == [1, 4]
assert indices_of_vowels("") == []
assert indices_of_vowels("aeiou") == [0, 1, 2, 3, 4]
```

Write a function – solution

```
def indices_of_vowels(string):
  result = [] # initialize empty list to hold indices
  index = 0 # initialize index
  while index < len(string):
    if string[index] in "aeiou": # check if character is vowel
      result.append(index) # append index to result
    index += 1 # increment index

  return result

def main():
  assert indices_of_vowels("hello") == [1, 4]
  assert indices_of_vowels("") == []
  assert indices_of_vowels("aeiou") == [0, 1, 2, 3, 4]
  print("Passed all tests.")

main()
```

```
Passed all tests.
```

Write a function

1. Its name is `reverse_list`
2. It takes a list as argument
3. It returns a new list with the items of the original list inverted

Test case:

```
test_strings = ["banana", "apple", "grape"]
assert reverse_list(test_strings) == ["grape", "apple", "banana"]
```

Write a function – solution

```
def reverse_list(items):
  index = len(items) - 1 # initialize index
  inverted_list = []
  while index >= 0:
```

```python
    inverted_list.append(items[index])
    index -= 1
  return inverted_list

def main():
  test_strings = ["banana", "apple", "grape"]
  assert reverse_list(test_strings) == ["grape", "apple", "banana"]
  print("Passed test")

main()
```

```
Passed test
```