Today's Topics:

- For loops
- Enumerate
- Looping practice

Lee wilkins

Office : 6C.9

Contact: MIO

INTRODUCTION TO COMPUTER PROGRAMMING IN ENGINEERING AND SCIENCE

Comp Sci assignments (All tests and assignments occur Wednesdays in our lab block)

Test 1 week 7 (15%) Wednesday March 5)

Assignment 2 week 8 (10%)

Assignment 3 week 11 (10%)

Test 2 week 13 (15%)

Physics assignments

Assignments (4 x 2%) 8% Date communicated by the Physics teacher

Project 1: Solving differential equations 10% Week 11

Project 2: Applying programming in science 22% Week 15

GRADE BREAKDOWN REVIEW

```
num_list = ["339", "362", “859"]
```

Add an element at a specific index

```
num_list.insert(2, “test")
print(num_list)
# ['339', '362', 'test', '859']
```

Remove an element at a specific index

```
num_list.pop(2)
print(num_list)
# ['339', '362', 'test', '859']
```

Reverse the list

```
num_list.reverse()
print(num_list)
# ['859', '362', ‘339']
```

Add to the end of the list

```
num_list.append("TESTING!")
print(num_list)
# ['859', '362', '339', 'TESTING!']
```

MODIFYING A LIST WITH A LIST METHOD

Lists need to be defined as a data type before you can perform list methods on them. For example:

```
 new_items.append("Add this to the list”)
NameError: name 'new_items' is not defined
```

```
new_items = "First element”
new_items.append("Add this to the list”)
AttributeError: 'str' object has no attribute ‘append’
```

```
new_items = []
new_items.append("Add this to the list”)
```

By defining an empty list, you have access to list methods.

CREATING LISTS

For: Loops through a range of items, or for each item in a set of items

While: Loops while a condition is true

LOOPS

For / in loop can be used to iterate through each item in a list or in a range.

```
for item in space
    #do task
```

LOOPS: FOR IN

For / in gives us the value of each element in the list

```
for animal in animals:
    print(animal, end=", “)
> lions, tigers, bears,
```

Range gives us the number within a range (here, the range is the length of the animals list, but this can be any range.

```
for animal in range(len(animals)):
    print(animal, end=", “)
# 0, 1, 2,
```

Enumerate gives us the index and value for each element in the list

```
for i, animal in enumerate(animals):
    print(f"{i} index contains {animal}”)

0 index contains lions
1 index contains tigers
2 index contains bears
```

LOOPS: FOR, RANGE, ENUMERATE

Enumerate is used to access both the index and value of each element. This is useful if we need to know the exact position within a list, not only the value. T

```
animals = ["lions", "tigers", "bears"]
for i, animal in enumerate(animals):
    print(f"{i} index contains {animal}")
```

You can use enumerate with an index in the for loop, or without. Without you will need to access each element individually.

```
for animal in enumerate(animals):
    print(animal)
    print(animal[0])
    print(animal[1])
```

```
(0, 'lions') 0 lions
(1, 'tigers') 1 tigers
(2, 'bears') 2 bears
```

LOOPS: FOR ENUMERATE

When modifying a list in a loop, you must call it by the list index and not the variable in the for loop.

```
for animal in animals:
    animal.upper()
print(animals)
#['lions', 'tigers', 'bears']
```

You can display or assign elements in a list like this, but you can't use this to modify the list because animals is a a different variable

```
for index, animal in enumerate(animals):
    animals[index] = animals[index].upper()
print(animals)
# ['LIONS', 'TIGERS', 'BEARS']
```

Here we are using the index of the list to modify the element

MODIFYING A LIST IN A LOOP

It may be a good idea to create two lists and store each if you are sorting elements

```python
numbers_to_sort = [34,36,1,77,352,485]
even_nums = []
odd_nums = []
for nums in numbers_to_sort:
    if nums%2:
        odd_nums.append(nums)
    else:
        even_nums.append(nums)
print(f"Even nums are {even_nums} and odd nums are {odd_nums}")
```

CREATING LIST IN A LOOP

There are any ways to search lists and strings, depending what you are looking for and at what level you want to understand your data. Here are some examples.

```python
animals = ["lions", "tigers", "bears", "lions", "ducks"]
find = "lions"

if find in animals: print("its there")
# Checks if the item is present anywhere in the list
print(animals.index(find))
#Returns the exact location of the first instance

find_counter = 0
for animal in animals:
    if animal == find:
        find_counter+=1
print(find_counter)
# Find how many times it appears

find_locations = []
for index, animal in enumerate(animals):
    if animal == find:
        find_locations.append(index)
print(f"The word {find} appears {len(find_locations)} times at {find_locations}")
# Find how many times and where it appears in the list
```

EXAMPLE: WAYS TO FIND

Range returns a value that can be used to move through a set number of iterations

Here range is 0 - 6 (non inclusive)

```
for x in range(6):
  print(x)
# 0 1 2 3 4 5
```

Here range is 3 - 6 (non inclusive)

```
for x in range(3, 6):
  print(x)
# 3 4 5
```

LOOPS: FOR IN RANGE

With three values we can set the increment per loop, here I'm setting it to 2. So it will count by 2 (non inclusively)

```
for x in range(0, 6, 2):
  print(x, end=“ ")
# 0 2 4
```

Note: you can use end=“, “ to indicate how the line ends instead of a new line. Ex; here we put a “, “ instead of a line break. This Is useful when debugging loops!

```
for i in range(-1, 5, 2):
    print(i, end=", ") # prints: -1, 1, 3,
```

LOOPS: FOR IN RANGE

```
# Find a range in a string
my_string = "A string to iterate through, lets find some letters"
start_position = my_string.find("string")
end_position = my_string.find("find")
# r = range(start_position, end_position)
for i in range(start_position, end_position):
    print(my_string[i], end= "  “)
```

This example uses range and for loop in a string.

We are using range to find the indexes in a string between two words and listing each letter between them. We can also start at the end of string by adding the length of the index of the word.

```
start_position = my_string.find("string")+len("string")
```

LOOPS: FOR IN RANGE

Fruit represents a single item in the list. It changes as we iterate through the list. Every loop, we're looking at the next item of fruit inside fruits.

```
fruits = ['apple', 'banana',
'cherry']
```

Fruits references the specific list

```
for fruit in fruits:
    print(len(fruit))
```

Fruit can be named anything, but this is typical naming convention.

FOR / IN LOOPS

Before we can really see the power of for loops, we need to talk about lists.

Lists are a way of storing many things in a single variable. You can access them like we do with string indexes, remembering 0 is the first item in a list. A list can be many o

```
my_string_list = ["apple", "oranges", "bananas"]
print(my_string_list[0]) # apples
my_int_list = [2, 3, 10]
print(my_int_list[1]) #3
my_float_list = [2.4, 502.4, 2.5]
print(my_float_list[2]) #2.5
my_list_list = [ [1,4,5], [3,5,4], [4,2,5]]
print(my_list_list[1][1]) #5
```

LISTS

Lists can contain multiple data types. List is the entire structure (with its own methods) and each item can be accessed and on its own. Its important to pay attention to data types if your list is like this!

```
my_mixed_list = [2.4, "502.4", 2]
print(type(my_mixed_list)) # <class 'list'>
print(type(my_mixed_list[1])) # <class ‘str'>
```

LISTS

Similar to strings, there are list methods. You can find them here https://www.w3schools.com/python/python_ref_list.asp

List methods can only be performed on variables that have the data type list.

```
fruits = ['apple', 'banana', 'cherry']
print(fruits) # ['apple', 'banana', 'cherry']
fruits.reverse()
print(fruits)# ['cherry', 'banana', 'apple']
```

LISTS METHODS

Append will add a new item to the list (to the end)

```
new_fruit = input("What is another fruit?")
fruits.append(new_fruit)
print(fruits)
```

LISTS: APPEND

We can turn a string into a list if it has a delineator (something to separate the items of a list.

```
# Convert a string into a list
my_string_to_convert = "apples, oranges, bananas"
print(my_string_to_convert) # apples, oranges, bananas
my_string_to_convert = my_string_to_convert.split(“,") # ['apples', '
oranges', ' bananas’]
print(my_string_to_convert)
```

This string is split by the comma, but it has spaces! We can handle this two ways:

- Make a string that has no spaces (“apples, oranges, bananas”) or trip the white space with replace

```
my_string_to_convert = my_string_to_convert.replace(" ", "")
print(my_string_to_convert)
```

STRING TO LIST

Delineators can be anything! “this|is|my|string” or even “this is my string” where the delineator is a space.

This can be useful when getting data from a larger file that ou need to clean up. For example, data from a study or collection!

STRING TO LIST

LAB TIME

Coming up: Nested loops, matplotlib and Jupyter notebook

NEXT CLASS: