lists (class slides)

CSc 110 - Lists

Sequences

We've worked with `strings`, which are sequences in Python. Sequences can be indexed with `[ ]`:

```
word = "uncopyrightable"
word[0] # returns first character in string
```

```
'u'
```

```
word[1] # returns second character in string
```

```
'n'
```

We also have `lists` in Python, which are also sequences and can indexed with `[ ]`:

```
numbers = [4, 2, 5, 7]
numbers[0] # returns first item in list
```

```
4
```

Data Structures

- Data structures is a way to organize data when coding
- Data structures allow for easy access and **modification** of values
- You can see data structures as a collection of data values
- In Python `lists` are a data structure that is:
  - mutable (you can change the values in it)
  - unordered/unsorted – you can have repeated elements in a list

Creating lists

```python
# empty list
no_numbers = []

# list of integers
numbers = [1, 5, 2, 10, 7]

# list of strings
names = ['ron', 'joe', 'kyle']

# mixed types list
values = [1, 1.15, 7, 1.75, 'those']
```

Evaluate the expressions

```python
numbers = [2, 3, 2, 4, 5]
numbers[1] + numbers[4]

numbers = [2.0, 3, 1.3, 4]
numbers[0] * numbers[2]

words = ['the', 'bear', 'in', 'the', 'tree']
words[3] + words[4] + words[1]

floats = [1.2, 3.4, 0.3, 1.0, 3.2]
len(floats)
```

Evaluate the expressions

```python
numbers = [2, 3, 2, 4, 5]
numbers[1] + numbers[4]
```

```
8
```

```python
numbers = [2.0, 3, 1.3, 4]
numbers[0] * numbers[2]
```

```
2.6
```

```python
words = ['the', 'bear', 'in', 'the', 'tree']
words[3] + words[4] + words[1]
```

```
'thetreebear'
```

```python
floats = [1.2, 3.4, 0.3, 1.0, 3.2]
len(floats)
```

```
5
```

Write a function

1. Its name is `sum_all`
2. It takes a list of `numbers` as an argument
3. It runs a loop that iterates through the values in `numbers` summing all values (HINT: you need to create a variable that will aggregate or accumulate the sum)
4. It returns the sum of all values in `numbers`

5. Use `while` (define an `index` before the loop, use index in the `while` condition, change the `index` inside the loop)

```python
assert sum_all([2, 2, 2]) == 6
assert sum_all([1, 2, 1, 1]) == 5
assert sum_all([]) == 0
```

Write a function – solution

Add test cases to the solution below:

```
def sum_all(numbers):
  total = 0
  index = 0
  while index < len(numbers):
    total += numbers[index]
    index += 1

  return total

def main():
  assert sum_all([2, 2, 2]) == 6
  assert sum_all([1, 2, 1, 1]) == 5
  assert sum_all([]) == 0

main()
```

Submit code for attendance

Submit your `sum_all` function to Gradescope for attendance.

Name your file `sum_list.py`

Loop Table

```
sum_all([2, 1, 5, 2, 3])
```

index	index < len(numbers)	numbers[index]	total
0	True	2	$0 + 2 = 2$
1	True	1	$2 + 1 = 3$
2	True	5	$3 + 5 = 8$
3	True	2	$8 + 2 = 10$
4	True	3	$10 + 3 = 13$
5	False	-	13

Loop Table

```
sum_all([2, 1, 3, 4])
```

index	index < len(numbers)	numbers[index]	total
|			
|			
|			
|			
|			
|			

Loop Table – solution

```
sum_all([2, 1, 3, 4])
```

index	index < len(numbers)	numbers[index]	total
0	True	2	2
1	True	1	3
2	True	3	6
3	True	4	10
4	False	-	10

Python Tutor

You can also visualize code in python on Python Tutor

Using a control variable

Remember how to get the max of three numbers?

```
def max3(x, y, z):
  max = x # assume max is first number

  if y > max:
    max = y # assumption is incorrect, assume y is max
  if z > max:
    max = z # assumption is incorrect, z is max

  return max

def main():
  print( max3(1, 2, 2) ) # 2

main()
```

```
2
```

Activity: adapt this function (`max_list`) to take a list of numbers instead of three numbers.

Max of list – solution

```
def max_list(numbers):
  '''
  Given a list of number, this function returns the highest number.
  Args:
    List of numeric values
  Returns:
    Max (float or integer, whatever value type is the highest)
  '''
  max = numbers[0]
  index = 1
  while index < len(numbers):
    if numbers[index] > max:
      max = numbers[index]
    index += 1
  return max

def main():
  print( max_list([1, 2, 2, 1, 3, 1, 1]) ) # 3
  print( max_list([3, 2, 2, 1, 0, 1, 1]) ) # 3

main()
```

```
3
3
```

Loop table

```
max_list([2, 1, 3, 1])
```

index	index < len(numbers)	numbers[index]	max
|			
|			
|			
|			
|			
|			

Loop table – solution

```
max_list([2, 1, 3, 1])
```

index	index < len(numbers)	numbers[index]	max
1	True	1	2
2	True	3	3
3	True	1	3
4	False	-	3

Max solution

- What about empty lists?
- How to get the min instead?

Write a function

1. Its name is `double`
2. It takes a list of numeric variables as argument
3. It iterates over the list (use `while`) doubling (multiplying by two) each value in the list
4. It returns the modified list

```
assert double([0, 1, 2, 3]) == [0, 2, 4, 6]
```

Write a function – solution

```
def double(numbers):
  index = 0
  while index < len(numbers):
    numbers[index] *= 2
    index += 1
  return numbers

def main():
  original_list = [0, 1, 2, 3]
  new_list = double(original_list)
  assert original_list == new_list
  assert original_list == [0, 2, 4, 6]

  print("Passed all tests")

main()
```

Submit attendance

Name your file `double.py` and submit it to gradescope.

Slicing lists

- Range – `list[2:4]`
- Whole list – `list[:]`
- Everything but last character – `list[:-1]`

The same slicing can be done with strings