CompSci 101

range function
KISS Principle

- Think of the non-computing context for any word/terms
- KISS model
  - Work smarter, not harder!!
- “Good programmers are simply good designers.”
  - *Dr. Washington*
- Design first and always!
- Importance of reusability
- **USE PYTHON TUTORS IF YOU HAVE QUESTIONS!**
for loops

```
if __name__ == '__main__':
    for number in [0, 1, 2, 3]:
        print(number)
```

- What about larger numbers?

- `range(stop)`
  - 0 up to (not including) stop

- `range(start, stop)`
  - Specify start value (increment by 1)

- `range(start, stop, step)`
  - Specify step value
CompSci 101
Accumulators
Why use loops?

• Repetition
  • Keeping a running total (counter)
  • Summing (other repetitive calculations)

• Accumulators
  • “Accumulate”- *acquire an increasing number of quantity of.*

• Rules for accumulators
  • Must initialize the “running total”
  • Must not initialize “inside the loop”
  • Accumulator must increase the total with each loop iteration
Example: 6.5-Accumulator Pattern

def square(x):
    runningtotal = 0
    for counter in range(x):
        runningtotal = runningtotal + x

    return runningtotal

if __name__ == '__main__':
    toSquare = 10
    squareResult = square(toSquare)
    print("The result of", toSquare, "squared is", squareResult)
Another way to use accumulators

def square(x):
    '''raise x to the second power'''
    runningtotal = 0
    for counter in range(x):
        runningtotal = runningtotal + x
    return runningtotal

def square(x):
    '''raise x to the second power'''
    runningtotal = 0
    for counter in range(x):
        runningtotal += x
    return runningtotal
CompSci 101
Traversing and accumulating strings
Print each character in a string

```python
if __name__ == '__main__':
    name = 'Tiana'
    for i in range(5):
        print(name[i])
```

Can this be simplified?
What about printing the characters in reverse order?
Accumulators with Strings

• How is “+” used with strings?
  • Concatenation
  • result = “string1” + “string2”

• Still require initialization
  • Empty string (“”) instead of 0

• Still “acquiring/increasing quantity.”
Example: 9.4-Accumulator Patterns with Strings

def removeVowels(s):
    vowels = "aeiouAEIOU"
    sWithoutVowels = ""
    for eachChar in s:
        if eachChar not in vowels:
            sWithoutVowels = sWithoutVowels + eachChar
    return sWithoutVowels

print(removeVowels("compsci"))
print(removeVowels("aAbEeflijOopUus"))
Why using “not in” instead of “in”?

• KISS

• Which is simpler to use?
  • What’s required to use “in”?
  • What’s required to use “not in”?
  • Which is simpler to design/implement?
CompSci 101
Traversing lists
Which is better to traverse list?

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

for position in range(len(fruits)):  # by index
    print(fruits[position])

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

for afruit in fruits:  # by item
    print(afruit)
```
Remember lists are mutable...

numbers = [1, 2, 3, 4, 5]
print(numbers)

for i in range(len(numbers)):
    numbers[i] = numbers[i] ** 2

print(numbers)