

## CompSci 101

### *range* function

1

## 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 TUTORIALS IF YOU HAVE QUESTIONS!**

2

## 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

3

4

## CompSci 101 Accumulators

5

### 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

6

### 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)
```

7

### Another way to use accumulators

def square(x):	def square(x):
"""raise x to the second power"""	"""raise x to the second power"""
runningtotal = 0	runningtotal = 0
for counter in range(x):	for counter in range(x):
runningtotal = runningtotal + x	<b>runningtotal += x</b>
return runningtotal	return runningtotal

8



9

## CompSci 101

### Traversing and accumulating strings

10

Print each character in a string

```
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?

11

### Accumulators with Strings

- How is "+" used with strings?
  - Concatenation
  - result = "string1" + "string2"
- Still require initialization
  - Empty string ("") instead of 0
- Still "acquiring/increasing quantity."

12

### 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"))
```

13

### 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?

14



15

CompSci 101  
Traversing lists

16

### Which is better to traverse list?

```
fruits = ["apple", "orange", "banana", "cherry"]    fruits = ["apple", "orange", "banana", "cherry"]

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

17

### 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)
```

18



19