Accumulators

Today's Plan

- Review
- More examples of **`for`** loops
- String formatting
- The accumulator pattern

Review: program structure

```
"""
This is a multi-line comment in which
I describe what my program does.

Pia Python, CS 21
"""

# Define block of code called 'main'
def main():
    # Indented lines of code
    ...


# Invoke main (unindented)
main()
```

Review: assignment

```
"""
Program that uses re-assignment, for loops.

Sy Burrspace, CS 21
"""

def main():
    a = 2
    b = 4
    b = b + 1
    b = b + a
    print(b)

main()
```

Review

- A **list** is an ordered, indexed collection of values
- Creating lists
  - `L = [3, 5, 10]`
  - `L = range(1, 4)`
- Lists and strings are both **sequences**
- The `**` operator does exponentiation

Review

- **`for`** loops let you repeat lines of code, one repetition for each item in a sequence

- **`for`** loop syntax:

```
for <variable> in <sequence>:
    <block>
```

- We say that a for loop **traverses** or **iterates** over a sequence. The variable in a for loop is an **iteration variable**

```
for item in ["apples", "ice cream", "bread"]:
    print()
    print("item is: " + item)
```

- Output:

```

item is: apples

item is: ice cream

item is: bread
```

```
for item in ["apples", "ice cream", "bread"]:
    print()
    print("item is: " + item)
```

- is the same as -

```
item = "apples"
print()
print("item is: " + item)

item = "ice cream"
print()
print("item is: " + item)

item = "bread"
print()
print("item is: " + item)
```

Four for loop types

- Perform all instructions in the for loop body…

  …for each item in a list

  …for each item in a list produced by `range`

  …for each character in a string

  …`n` times

```
for i in range(1, 4):
    square = i**2
    message = str(i) + " squared is " + str(square)
    print(message)
```

Output:

```
1 squared is 1
2 squared is 4
3 squared is 9
```

String formatting

```
for i in range(1, 4):
    square = i**2
    message = str(i) + " squared is " + str(square)
    print(message)
```

- Can be written more concisely as:

```
for i in range(1, 4):
    print("%d squared is %d" % (i, i**2))
```

String formatting

- `<format string> % value`
- `<format string> % (value1, value2, …)`
- `%s, %d, %f` are **placeholders** for strings, ints, floats, respectively

```
>>> "====%s====" % "CS 21"
'====CS 21===='
>>> "====%s====" % "ululate"
'====ululate===='
>>> i = 3
>>> "%d squared is %d" % (i, i**2)
'3 squared is 9'
```

Accumulator pattern

- Use this pattern when you want to *combine the values in a sequence into one value*

Accumulator Pattern

- Initialize accumulator, a variable
  - *What should the initial value be?*
- Inside of a loop, update accumulator to new value, using old value
  - *What operation do I use to do update?*
- Use final value after the loop finishes
  - *What do I do with it?*

See you Friday!