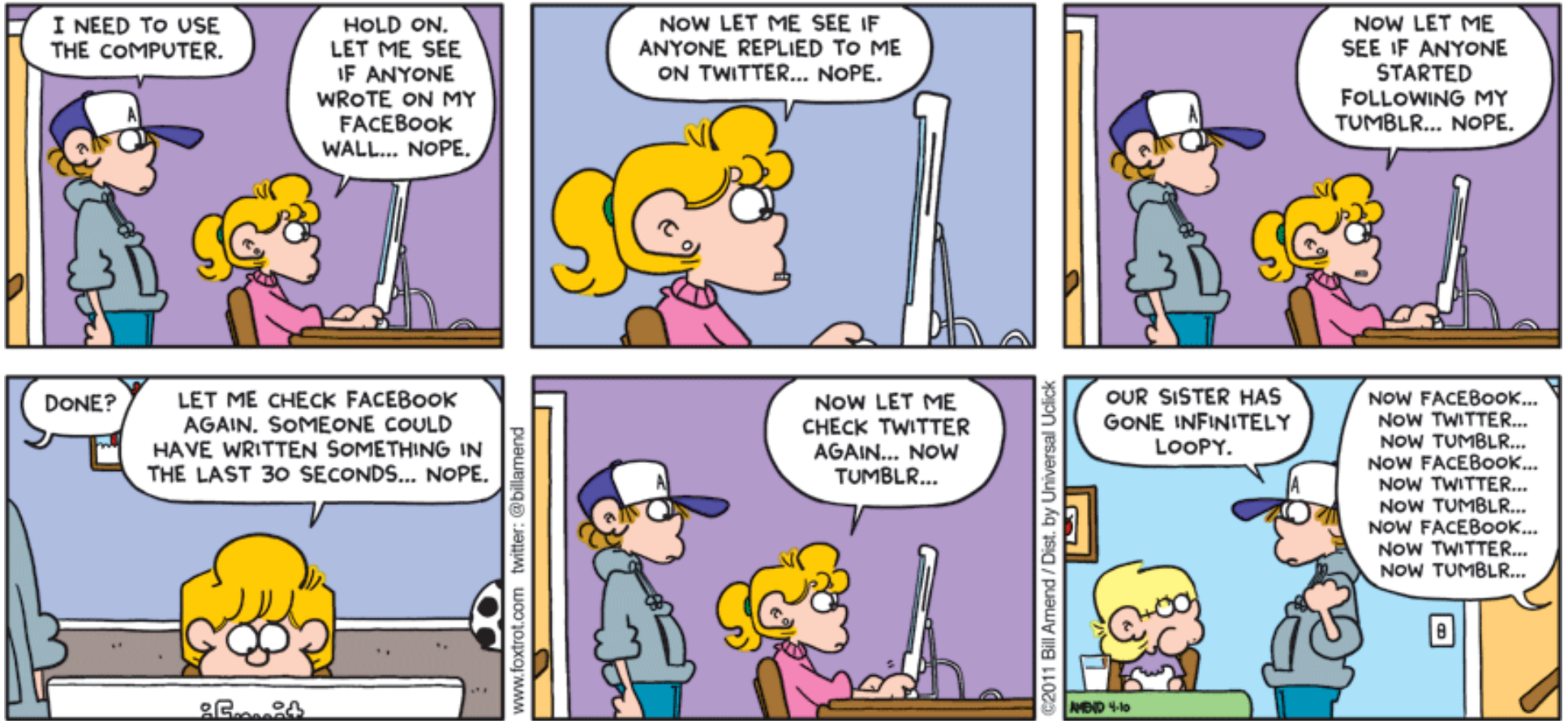
CSc 110, Autumn 2017

Lecture 5: The `for` Loop and user input

Adapted from slides by Marty Stepp and Stuart Reges



Declaration and assignment

- **variable declaration and assignment**:

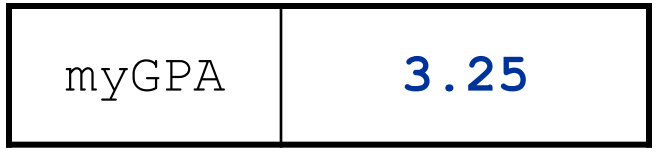
  Sets aside memory for storing a value and stores a value into a variable.

  - Variables must be declared before they can be used.
  - The value can be an expression; the variable stores its result.

- Syntax:

  name = **expression**

```
zipcode = 90210

myGPA = 1.0 + 2.25
```



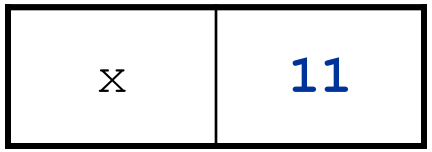
Using variables

- Once given a value, a variable can be used in expressions:

```
x = 3                   # x is 3
y = 5 * x - 1           # now y is 14
```

- You can assign a value more than once:

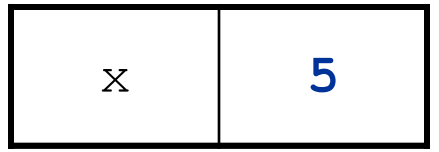
```
x = 3                   # 3 here

x = 4 + 7               # now x is 11
```



Assignment and algebra

- Assignment uses = , but it is not an algebraic equation.
  - = means, *"store the value at right in variable at left"*
    - The right side expression is evaluated first, and then its result is stored in the variable at left.
- What happens here?

```
x = 3
x = x + 2    # ???
```



Receipt question

Improve the receipt program using variables.

```
def main():
    # Calculate total owed, assuming 8% tax / 15% tip
    print("Subtotal:")
    print(38 + 40 + 30)

    print("Tax:")
    print((38 + 40 + 30) * .08)

    print("Tip:")
    print((38 + 40 + 30) * .15)

    print("Total:")
    print(38 + 40 + 30 + (38 + 40 + 30) * .15 + (38 + 40 + 30) * .08)
```

Getting rid of repetition

- Functions
- Variables
- String Multiplication
  - Allows you to print multiple occurrences of the same string without typing them all out

```
print("meow" * 3)          # meowmeowmeow
```

- What if you want to repeat function calls?

Repetition with `for` loops

- So far, repeating an action results in redundant code:

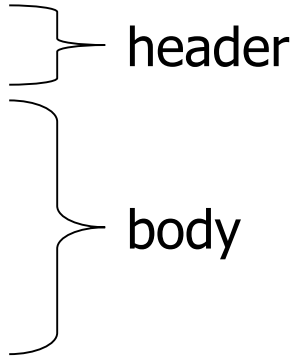
```
make_batter()
bake_cookies()
bake_cookies()
bake_cookies()
bake_cookies()
bake_cookies()
frost_cookies()
```

- Python's **`for` loop** statement performs a task many times.

```
mix_batter()
for i in range(1, 6):     # repeat 5 times
    bake_cookies()
frost_cookies()
```

`for` loop syntax

```
for variable in range (start, stop):
    statement
    statement
    ...
    statement
```



- Set the variable equal to the start value
- Repeat the following:
  - Check if the **variable** is less than the stop. If not, stop.
  - Execute the **statement**s.
  - Increase the variable's value by 1.

Control structures

- **Control structure**: a programming construct that affects the flow of a program's execution
- Controlled code may include one or more statements
- The `for` loop is an example of a looping control structure

Repetition over a range

```
print("1 squared =", 1 * 1)
print("2 squared =", 2 * 2)
print("3 squared =", 3 * 3)
print("4 squared =", 4 * 4)
print("5 squared =", 5 * 5)
print("6 squared =", 6 * 6)
```

- Intuition: "I want to print a line for each number from 1 to 6"

- The `for` loop does exactly that!

```
for i in range(1, 7):
    print(i, "squared = ", i * i)
```

- "For each integer **i** from 1 through 6, print ..."

Loop walkthrough

```
for i in range(1, 5):
    print(i, "squared =", i * i)

print("Whoo!")
```

Output:

```
1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
Whoo!
```

Multi-line loop body

```
print("+----+")
for i in range(1, 4):
    print("\\    /")
    print("/    \\")
print("+----+")
```

- Output:

```
+----+
\    /
/    \
\    /
/    \
\    /
/    \
+----+
```

Expressions for counter

```
high_temp = 5
for i in range(-3, high_temp // 2 + 1):
    print(i * 1.8 + 32)
```

- Output:

```
26.6
28.4
30.2
32.0
33.8
35.6
```

Rocket Exercise

- Write a method that produces the following output:

```
T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
blastoff!
The end.
```

`print('', end='')`

- Adding `, end=''` allows you to print without moving to the next line
  - allows you to print partial messages on the same line

```
high_temp = 5
for i in range(-3, high_temp // 2 + 1):
    print(i * 1.8 + 32, end=' ')
```

  - Output:

```
26.6  28.4  30.2  32.0  33.8  35.6
```

    - Either concatenate `'  '` to separate the numbers or set `end='  '`

Changing step size

- Add a third number to the end of range, this is the step size
  - A negative number will count down instead of up

```
print("T-minus ")
for i in range(10, 0, -1):
     print(str(i) + ", ", end="")
print("blastoff!")
print("The end.")
```

  - Output:

```
T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!
The end.
```

Constants

- **constant**: A fixed value visible to the whole program.
  - value should only be set only at declaration; shouldn't be reassigned

- Syntax:
  - Just like declaring a normal variable:

    name = **value**

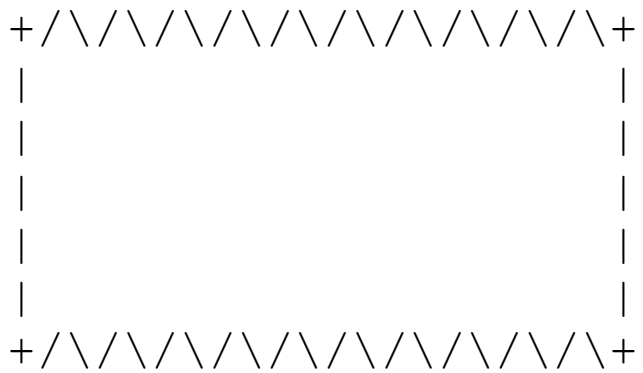
  - name is usually in ALL_UPPER_CASE

  - Examples:

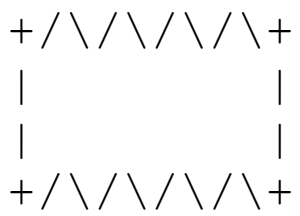
```
DAYS_IN_WEEK = 7
INTEREST_RATE = 3.5
SSN = 658234569
```

Constants and figures

- Consider the task of drawing the following scalable figure:



Multiples of 5 occur many times



The same figure at size 2

Constant tables

```
SIZE = ...
```

- What equation would cause the code to print:

```
2 7 12 17 22
```

- To see patterns, make a table of `SIZE` and the numbers.
  - Each time `SIZE` goes up by 1, the number should go up by 5.
  - But `SIZE * 5` is too great by 3, so we subtract 3.

`SIZE`	number to print	`5 * SIZE`	`5 * SIZE - 3`
1	2	5	2
2	7	10	7
3	12	15	12
4	17	20	17
5	22	25	22

Constant tables question

- What equation would cause the code to print:

```
17 13 9 5 1
```

- Let's create the constant table together.
  - Each time `SIZE` goes up 1, the number printed should ...
  - But this multiple is off by a margin of ...

`SIZE`	number to print	`-4 * SIZE`	`-4 * SIZE+ 21`
1	17	-4	17
2	13	-8	13
3	9	-12	9
4	5	-16	5
5	1	-20	1

Interactive programs

interactive program: Reads input from the console.

- While the program runs, it asks the user to type input.
- The input typed by the user is stored in variables in the code.

- Can be tricky; users are unpredictable and misbehave.
- But interactive programs have more interesting behavior.

`input`

- **`input`**: An function that can read input from the user.

- Using an `input` object to read console input:

  name `= input(`**prompt**`)`

  - Example:

    ```
    name = input("type your name: ")
    ```

    - The variable `name` will store the value the user typed in

input example

```
def main():
    age = input("How old are you? ")

    years = 65 - age
    print(years, " years until retirement!")
```

age | 29

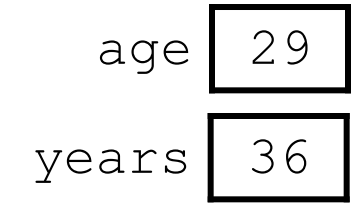
- Console (user input underlined):

```
How old are you? 29
```

```
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    print(65 - age)
TypeError: unsupported operand type(s) for -:
'int' and 'str'
```

input example

```
def main():
    age = int(input("How old are you? "))

    years = 65 - age
    print(years, "years until retirement!")
```



- Console (user input underlined):

```
How old are you? 29
36 years until retirement!
```