

CMSC201

Computer Science I for Majors

Lecture 07 – While Loops (cont)

Last Class We Covered

- Using **while** loops
 - Syntax of a **while** loop
 - Interactive loops
 - Infinite loops and other problems
- Practice with **while** loops

Any Questions from Last Time?

Today's Objectives

- To learn about constants and their importance
- To explore more and different **while** loops
 - Sentinel loops
 - Boolean flags
- To get more practice with **while** loops

Constants (and Magic)

Literal Values

- Remember, *literal values* are any integer, float, or string that is *literally* in the code
- Literals sometimes have a specific meaning
 - The strings “no” or “yes” as valid user choices
 - Having 7 days of the week, or 12 months in a year
- The meaning of these literals can be difficult to figure out as the program gets longer, or as you work with code you didn't write

Literal Value Confusion

- What do the pieces of code below do/mean?

```
num = int(input("Enter a number (1 - 52): "))
```

```
if choice == 4:  
    print("Thanks for playing!")
```

```
while year < 1900 or year > 2017:  
    print("Invalid choice")
```

Literal Value Confusion

- What do the pieces of code below do/mean?

```
num = int(input("Enter a number (1 - 52): "))
```

Weeks in a year? Cards in a deck?

```
if choice == 4:
```

```
    print("A menu option? To quit the program?
```

```
while year < 1900 or year > 2017:
```

```
    print("Invalid cho. Inputting a valid year? For what?
```


Literals are Magic!

- These literal values are “magic”, because their meaning is often unknown
 - Called magic numbers, magic strings, etc.
- Other problems include:
 - Reason for choosing the value isn’t always clear
 - Increases the opportunity for errors
 - Makes the program difficult to change later
 - Which 52 is weeks, and which is size of a card deck?



Constants

- Instead of using “magic” literal values, replace them in your code with named ***constants***
- Constants should be ALL CAPS with a “_” (underscore) to separate the words
 - This follows CMSC 201 Coding Standards
- Having a variable name also means that typos will be caught more easily

Literal Value Clarification

- After using constants, the code might look like:

```
MENU_QUIT = 4      # more options listed above
```

```
MIN_YEAR  = 1900
```

```
MAX_YEAR  = 2017
```

```
if choice == MENU_QUIT:
```

```
    print("Thanks for playing!")
```

```
while birthYear < MIN_YEAR or birthYear > MAX_YEAR:
```

```
    print("Invalid choice")
```

“Magic” Numbers Example

- You’re looking at the code for a virtual casino
 - You see the number 21 `if value < 21:` ✘
 - What does it mean?
- Blackjack? Drinking age? VIP room numbers?
`if customerAge < DRINKING_AGE:` ✔
- Constants make it easy to update values – why?
 - Don’t have to figure out which “21”s to change

Another “Magic” Example

- Can also have “magic” characters or strings
 - Use constants to prevent any “magic” values
- For example, a blackjack program that uses the strings “H” for hit, and “S” for stay

```
if userChoice == "H":
```



```
if userChoice == HIT:
```



- Which of these options is easier to understand?
- Which is easier to update if it’s needed?

Using Constants

- Calculating the total for a shopping order

`MD_TAX` = `0.06` easy to update if tax rate changes

```
subtotal = input("Enter subtotal: ")
```

```
subtotal = float(subtotal)
```

```
tax = subtotal * MD_TAX
```

```
total = tax + subtotal
```

```
print("Your total is:", total)
```

we know exactly what this number is

Acceptable “Magic” Literals

- Not everything needs to be made a constant!
 - 0 and 1 as initial or incremental values
 - 100 when calculating percentages
 - 2 when checking if a number is even or odd
 - Numbers in mathematical formulas
 - `0.5*base*height` or `2*pi*(radius**2)`
- Most strings don't need to be constants
 - Only if the value has a meaning or specific usage

Constant Practice

- Which of these are fine as just literal values?

```
count = 0
```

```
count += 1
```

```
while count < 100:
```

```
if choice == 1:
```

```
if age < 0:
```

```
percent = num / 100
```

```
perimSquare = 4 * sideLen
```

```
print("Hello!")
```

```
while ans != "yes":
```


Constant Practice

- Which of these are fine as just literal values?

✓ `count = 0`

✓ `count += 1`

✗ `while count < 100:`

✗ `if choice == 1:`

✓ `if age < 0:`

✓ `percent = num / 100`

✓ `perimSquare = 4 * sideLen`

✓ `print("Hello!")`

✗ `while ans != "yes":`

Could argue that this should be `MIN_AGE`

If "yes" is used as an option through the whole program, this should be a constant

Where Do Constants Go?

- Constants go before `main()`, after your header comment

- All variables that aren't constants must still be inside of `main()`

```
# File:    hw2_part6.py
# Author:  Dr. Gibson
# etc...

MAX_DAYS = 30
WEEK_LEN = 7

def main():
    date = int(input("Please enter day: "))

    if date >= 1 and date <= MAX_DAYS:
        # etc...

main()
```

Are Constants Really Constant?

- In some languages (like C, C++, and Java), you can have a variable that CANNOT be changed
- This is not possible with Python variables
 - Part of why coding standards are so important
 - If code changes the value of **MAX_ENROLL**, you know that's a constant, and that it should not be changed

Sentinel Values and **while** Loops

When to Use `while` Loops

- `while` loops are very helpful when you:
 - Want to get input from the user that meets certain specific conditions
 - Positive number
 - A non-empty string
- Want to keep getting input until some “end”
 - User inputs a value that means they’re finished

what we’re
covering now

Sentinel Values

- *Sentinel values* “guard” the end of your input
- They are used:
 - When you don’t know the number of entries
 - In **while** loops to control data entry
 - To let the user indicate an “end” to the data
- Common sentinel values include:
 - **STOP**, **-1**, **0**, **QUIT**, and **EXIT**



Sentinel Loop Example

- Here's an example, where we ask the user to enter student names:

```
END = "QUIT"
```

```
def main():
```

```
    name = input("Please enter a student, or 'QUIT' to stop: ")
```

```
    while name != END:
```

```
        print("Hello", name)
```

```
        name = input("Please enter a student, or 'QUIT' to stop: ")
```

```
main()
```

Sentinel Loop Example

- Here's an example, where we ask the user to enter student names:

```
END = "QUIT"
```

sentinel values should be saved as a constant

```
def main():
```

initialize the loop variable with user input

```
name = input("Please enter a student, or 'QUIT' to stop: ")
```

```
while name != END:
```

check for the termination condition

```
print("Hello", name)
```

```
name = input("Please enter a student, or 'QUIT' to stop: ")
```

```
main()
```

get a new value for the loop variable

Sentinel Loop Example

- Here's an example, where we ask the user to enter student names:

```
END = "QUIT"
```

```
d
```

we'll cover how to actually use constants in an input string later

make sure to tell the user how to stop entering data

```
name = input("Please enter a student, or 'QUIT' to stop: ")
```

```
while name != END:
```

```
    print("Hello", name)
```

```
    name = input("Please enter a student, or 'QUIT' to stop: ")
```

```
main()
```

make sure to use the value before asking for the next one

Priming Reads

- This loop example uses a *priming read*
 - We “prime” the loop by reading in information before the loop runs the first time
- We duplicate the line of code asking for input
 - Once before the loop
 - And then inside the loop
- This is the preferred way to use sentinel loops

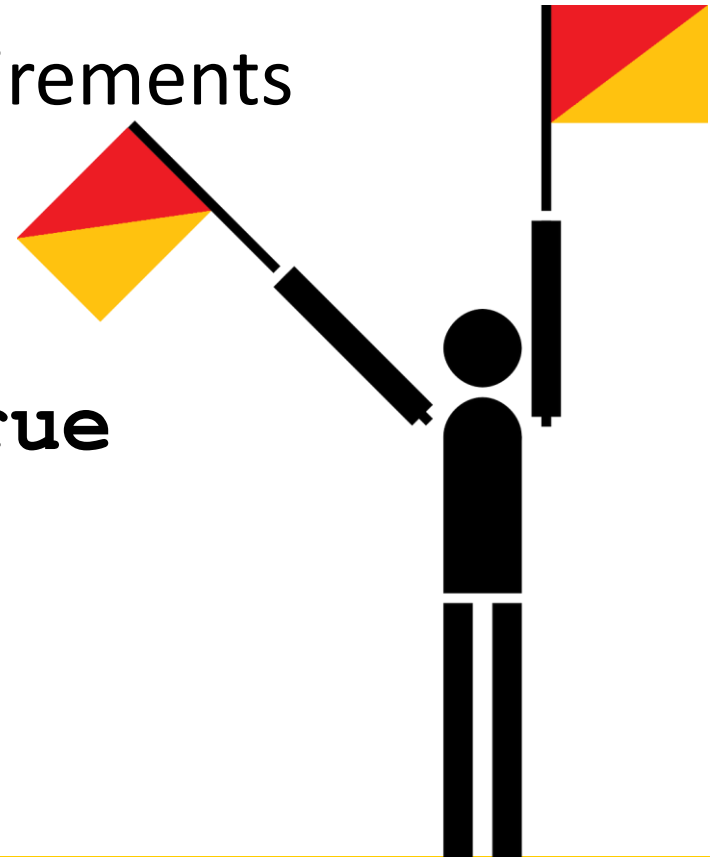
Boolean Flags

Complex Conditionals

- Sometimes, a **while** loop has many restrictions or requirements
 - Expressing them in one giant conditional is difficult, or maybe even impossible
- Instead, break the problem down into the separate parts, and use a single Boolean “flag” value as the loop variable

Boolean Flags

- A Boolean value used to control the while loop
 - Communicates if the requirements have been satisfied yet
- Value should evaluate to **True** while the requirements have not been met



General Layout – Multiple Requirements

- Start the **while** loop by
 - Getting the user's input
 - Assuming that all requirements are satisfied
 - (Set the Boolean flag so that the loop would exit)
- Check each requirement individually
 - For each requirement, if it isn't satisfied, change the Boolean flag so the loop repeats
 - (Optionally, print out what the failure was)

General Layout – Multiple Ways

- Start the **while** loop by
 - Getting the user's input
 - Don't assume the requirements have been met
 - (Do not change the Boolean flag at the start of the loop)
- Check each way of satisfying the requirements
 - If one of the ways satisfies the requirements, change the Boolean flag so the loop doesn't repeat

Boolean Flag Usage Examples

- Multiple requirements to satisfy
 - Password must be at least 8 characters long, no longer than 20 characters, and have no spaces or underscores
- Multiple ways to satisfy the requirements
 - Grade must be between 0 and 100, unless extra credit is allowed, in which case it can be over 100 (but still must be ≥ 0)

Time For...

LIVECODING!!!

Daily emacs Shortcut

- **Meta (M)**
 - Meta is another key, like “Control”
 - You can hold “Alt” down or you can hit “Esc”
 - “Alt” may not work on Macs
- **M+g+g**
 - Brings up a prompt at the bottom of the screen
 - Type in a number (and hit enter) to go to that line of your file

Announcements

- HW 2 is out on Blackboard now
 - Must have completed the Academic Integrity Quiz with a perfect score to see it
 - Due by Friday (February 23rd) at 8:59:59 PM
- Pre Lab Quizzes come out Fridays @ 10 AM
 - Nearly 20% of students aren't completing the quizzes each week

Image Sources

- Magic wand (adapted from):
 - https://commons.wikimedia.org/wiki/File:Magic_wand.svg
- Sentry guard (adapted from):
 - www.publicdomainpictures.net/view-image.php?image=160669
- Flag waver (adapted from):
 - <https://pixabay.com/p-34873>