Welcome to the CSC 110 midterm exam

Please log in to Canvas

Midterm exam

- 1 hour long
- Administered in Canvas
- If you have questions, raise your hand! I am happy to clarify when possible.

- Be respectful of your fellow students. If you finish early, please be quiet and let others finish.
- We will reconvene for lecture at 4:15

CSC 110: Week 6, Lecture 1

Dr. Bodine

Last week

- Decisions and Boolean logic
- Conditions
- If statements
- Exception handling

This week…

- Loops, loops and more loops!
- Definite loops
- Indefinite loops
- Sentinel loops
- Nested loops
- Boolean algebra

We've already been using loops

- Example: loop over elements in list.

```
for i in range(len(mylist)):
    <body>

for i in mylist:
    <body>
```

Counted loops

- Number of times to loop is well-defined
  - For example: by user input, by length of list, etc…

- Process executes a set number of times and then stops
  - Definite loop

```
for i in range(<number>):
    <body>
```

Definite loops

```
for <var> in <sequence>:
    <body>
```

- Automatically goes through the sequence
- We don’t need to do anything with <var> to get it to increment

Indefinite loops

- Iterates until a set of conditions is met
- Number of times body executes is not set ahead of time!
- Essentially waiting for a change of state of the condition

Indefinite loops in python

```
while <condition>:
      <body>
```

- Use while statements to define the condition and what to do while the condition is True
- Do not need to know the number of times it will execute
- Loops over body but does not automatically change anything
  - If using a loop variable, we have to increment it ourselves. This is different than definite loops where it automatically increments

Indefinite loop example

```
x = 0
while x < 11:
      print(x)
      x+=1
```

- First, we have to initialize a loop variable ourselves. What????
- Second, we have to increment it ourselves. Why????

While statements

- Pre-test loop
- Test the condition at the start of each loop
- We must update the condition within the body of the loop if we expect to end the loop

Think-pair-share

- Why would we want to use indefinite loops? Can you think of an example that we've already done in class where an indefinite loop would make it easier to achieve?

- What would happen for the following cases?

```
i = 0
while i>0:
      print(i)
      i = i+1

i = 1
while i>0:
      print(i)
      i = i+1
```

Infinite loops….

- Ooops!
- Happen when the condition can never change
  - i.e. when nothing can change the state of the condition
- Example, forgot to increment a loop variable

Common loop patterns

- Interactive loops
- Sentinel loops
- File loops
- Nested loops

Interactive loops

- User interacts with the program, for example, inputting numbers
- User signals when to end the loop
- Example, when taking a list of values to calculate the average

Let’s modify average from assignment 2:

```
#average test program
def main():
    ncounts = eval(input("Enter the number of elements in your list: "))
    mylist = eval(input("Enter your elements, separated by a comma: (x1,x2,...x_(n-1))"))

    sumtotal = 0 # accumulator variable

    #Loop over values to compute sum
    for i in range(ncounts):
        sumtotal = sumtotal + mylist[i]

    print("Average = ",sumtotal/ncounts)

main()
```

Modified version

```
#average test program with interactive loop
def main():
    sumtotal = 0 # accumulator variable for sum
    count = 0 #counting variable

    data = 'yes'
    while data == 'yes':
        number = eval(input("Enter a number:"))

        sumtotal = sumtotal + number
        count = count + 1
        data = input("Do you want to enter another number?")

    print("\nAverage = ",sumtotal/count)

main()
```

- Is this really all that much better?
- What happens if the person mistypes ‘yes’?
- Can we improve that?

Interactive loops

- This summing program isn't a great example, but it illustrates what we can do with this.
- Can also do the same thing with other user input
- Can check that processing went as planned

Boolean operators reminder

- Can make up complex expressions with operators
- <expression1> and <expression2>
- <expression1> or <expression1>
- not <expression>
  - True when <expression> is False. False when <expression> is true

In conditions

- Can use multiple conditions
- if (condition1 and condition2):
- while (condition 1 and condition2):
- Practice trying these out