CS177 Python Programming

Recitation 7 – Decision Structures
Announcement

• Midterm 1 is tonight 6:30pm – 7:30 pm CL50 224

• Make sure bring the PUID and a pencil
Table of Contents

• Review
  Logical operators
  Boolean type
  Decision structures
• Common mistakes when programming
Logical Operators

• Operator == checks if two values are equal or not; if yes, then condition becomes true.

• Ex: write an expression that evaluates to True if and only if variables *profits* and *losses* are exactly equal.
Logical Operators

• Operator `==` checks if two values are equal or not; if yes, then condition becomes true.

• Ex: write an expression that evaluates to True if and only if variables `profits` and `losses` are exactly equal.

  `profits == losses`
Logical Operators

• Operator != checks if the two values are equal or not; if values are NOT equal, then condition becomes true.

• Ex: Given the variable $c$, whose associated value is a string, write an expression that is True if and only if $c$ is not equal to a string consisting of a single blank.
Logical Operators

- Operator != checks if the two values are equal or not; if values are NOT equal condition becomes true.

- Ex: Given the variable c, whose associated value is a string, write an expression that is True if and only if c is not equal to a string consisting of a single blank.

  c != " "
Logical Operators

- Operator $<$ checks if the left value is less than the right value; if yes, then condition is true

- Ex: Working overtime is defined as having worked more than 40 hours during the week. Given the variable `hours_worked`, write an expression that evaluates to True if the employee worked overtime.
Logical Operators

• Operator < checks if the left value is less than the right value; if yes, then condition is true

• Ex: Working overtime is defined as having worked more than 40 hours during the week. Given the variable \( \text{hours\_worked} \), write an expression that evaluates to True if the employee worked overtime.

\[ 40 < \text{hours\_worked} \]
Logical Operators

• Operator > checks if the left value is greater than the right value; if yes, then condition is true

• Ex: Write an expression that evaluates to True if the value of index is greater than the value of last_index.
Logical Operators

• Operator > checks if the left value is greater than the right value; if yes, then condition is true

• Ex: Write an expression that evaluates to True if the value of index is greater than the value of last_index.

  \[ \text{index} > \text{last\_index} \]
Logical Operators

• Operator $\leq$ checks if the left value is less than OR equal to the right value; if yes, then condition is true

• Ex: Given the variables $number_{of\_men}$ and $number_{of\_women}$, write an expression that evaluates to True if the number of men is smaller than or equal to the number of women.
Logical Operators

• Operator $\leq$ checks if the left value is less than OR equal to the right value; if yes, then condition is true

• Ex: Given the variables $number\_of\_men$ and $number\_of\_women$, write an expression that evaluates to True if the number of men is smaller than or equal to the number of women.

   $number\_of\_men \leq number\_of\_women$
Logical Operators

• Operator $\geq$ checks if the left value is greater than OR equal to the right value; if yes, then condition is true.

• Ex: Write an expression that evaluates to True if $x$ is greater than or equal to $y$.
Logical Operators

• Operator >= checks if the left value is greater than OR equal to the right value; if yes, then condition is true.

• Ex: Write an expression that evaluates to True if $x$ is greater than or equal to $y$.
  
  $x \geq y$
Booleans

• Boolean expressions is an expression that can be evaluated as True or False.
• Examples we just went through.
  
  profits == losses
  c != " "
  hours_worked < 40
  index > last_index
  number_of_men <= number_of_women
  x >= y
  etc.
Boolean (AND)

• Logical operator **AND** results in true if both operands are true; otherwise the value is false

• Ex: Given the already defined variables `years_with_company` and `department`, write an expression that evaluates to True if `years_with_company` is less than 5 and `department` is not equal to 99.
Boolean (AND)

• Logical operator **AND** results in true if both operands are true; otherwise the value is false

• Ex: Given the already defined variables `years_with_company` and `department`, write an expression that evaluates to True if `years_with_company` is less than 5 and `department` is not equal to 99.

  `years_with_company < 5 and department != 99`
Boolean (OR)

• Logical operator OR results in true if at least operand is true; otherwise the value is false

• Ex: Given the already defined variables is_full_time_student and age, write an expression that evaluates to True if age is less than 19 or is_full_time_student is True.
Boolean (OR)

• Logical operator **OR** results in true if at least operand is true; otherwise the value is false

• Ex: Given the already defined variables `is_full_time_student` and `age`, write an expression that evaluates to True if `age` is less than 19 or `is_full_time_student` is True.

```
age < 19 or is_full_time_student
```
Boolean(NOT)

- Negation is a unary logical connective. It is normally takes truth to falsity and vice versa.

- Ex: Given the already defined variables age, write an expression that evaluates to True if age is not less than 21
Boolean(NOT)

• Negation is a unary logical connective. It is normally takes truth to falsity and vice versa.

• Ex: Given the already defined variables age, write an expression that evaluates to True if age is not less than 21

\[ \text{not(age < 21)} \]
Decision Structures

• Decision structures are statements that allow a program to execute different sequences of instructions for different cases, allowing the program to “choose” an appropriate course of action.
Decision Structures

• `if` statement is used to implement the decision

• Single if statement, two-way decisions and even multi-way decisions
Single if Statement

```
if <condition>:
    <body>
```
Single if Statement

- Given the variables `temperature` and `humidity`, write an code snippet that print a heat warning if and only if `temperature` is greater than 90 and `humidity` is less than 10.
Single if Statement

- Given the variables *temperature* and *humidity*, write an code snippet that print a heat warning if and only if *temperature* is greater than 90 and *humidity* is less than 10.

```python
if(temperature > 90 and humidity < 10):
    print(“Warning! Too hot!”)
```
Two-Way Decisions

```python
if <condition>:
    <body>
else:
    <body>
```

Diagram:

- **Discriminant (discrim < 0?):**
  - **No:** Calculate roots
  - **Yes:** Print "no roots"
Two-Way Decisions

• Write an if/else statement that compares age with 65, adds 1 to senior_citizens if age is greater than or equal to 65, and adds 1 to non_seniors otherwise.
Two-Way Decisions

• Write an if/else statement that compares \textit{age} with 65, adds 1 to \textit{senior\_citizens} if \textit{age} is greater than or equal to 65, and adds 1 to \textit{non\_seniors} otherwise.

\begin{verbatim}
if age >= 65:
    senior_citizens += 1
else:
    non_seniors += 1
\end{verbatim}
Multi-Way Decisions

- if <condition1>:
  <case1 statements>
elif <condition2>:
  <case2 statements>
elif <condition3>:
  <case3 statements>
...
else:
  <default statements>
Multi-Way Decisions

Flowchart:
- If `discrim < 0`, print "no roots".
- If `discrim >= 0`:
  - If `discrim == 0`:
    - Do Double Root
  - If `discrim > 0`:
    - Do Unique Roots
Multi-Way Decisions

• Write an if/else statement that adds 1 to minors if *age* is less than 18, adds 1 to *adults* if *age* is 18 through 64 and adds 1 to *seniors* if *age* is 65 or older.
Multi-Way Decisions

• Write an if/else statement that adds 1 to minors if age is less than 18, adds 1 to adults if age is 18 through 64 and adds 1 to seniors if age is 65 or older.

```python
if age < 18:
    minors += 1
elif age < 65:
    adults += 1
else:
    seniors += 1
```
For/While Loop

- The most commonly encountered mistake in the lab is the misusage of for/while loop.
- The following slides will discuss these two methods and show some examples.
For Loop

• *For* loops are traditionally used when you have a piece of code which you want to repeat a fixed number of times.

• There are two forms of for loop

  ```python
  for i in range(INT)
  for item in (list/string)
  ```
For Loop

• *for i in range(INT)* is used when the running times is known
• i.e., for loop from 0 to 2, therefore running 3 times:
  ```python
  for i in range(3):
  ```
• *for item in (list/string)* is often used when the list/string is known, i.e.,
  ```python
  string = "Hello World"
  for x in string:
      print x
  ```
While Loop

- As an alternative, there is the *while* loop. However, *while* is used when a condition is to be met, or if you want a piece of code to repeat forever.
While Loop

while True:
    x = eval(input("Enter a positive number: "))
    if x >= 0:
        x = math.sqrt(x)
        print("The square root of this number is: ", x)
    else:
        print("The number you entered was not positive!")
    break # Exit loop if number is invalid
String and List

• Both string and list are sequential.
• Each individual character/element can be accessed through indexing.
• The general form is <string>[<index>] or <list>[<index>].
• The index number always starts from 0
Functions

• Functions can be used to reduce code duplication, i.e. functions from the standard libraries

• Functions can make programs more easily understood and maintained, i.e. calculate final score for all CS177 students
Practice

• Use two variables $k$ and $total$ to write a $for$ loop to compute the sum of the squares of the first 50 counting numbers, and store this value $intotal$. Thus, your code should put $1*1 + 2*2 + 3*3 +... + 49*49 + 50*50$ into $total$. Use no variables other than $k$ and $total$. 
Practice

• Use the variables $k$ and $total$ to write a while loop that computes the sum of the squares of the first 50 counting numbers, and associates that value with $total$. Thus your code should associate $1^2 + 2^2 + 3^2 + \ldots + 49^2 + 50^2$ with $total$. Use no variables other than $k$ and $total$. 