CSC 171 - Introduction to Computer Programming

Lecture #2 - Decisions, Decisions: Boolean Expressions and Selection

What is a decision?

- A decision involves choosing whether to follow a given course of action, or which one of several different courses of action.
- We do this countless times during the day, making literally thousands of small decisions.
- For example, when we reach an intersection, we decide whether to cross it in the following manner:

```
IF the light is green
  THEN  cross the street
  ELSE  wait for the light to change
```

What is a decision? (continued)

- This decision has three parts:

the condition

IF the light is green

THEN cross the street ← *if the condition is true, we do this*

ELSE wait for the light to change

if the condition is false
we do this

The basic structure of decisions

- There are many decisions that do not have an IF-THEN-ELSE structure. For example,

  IF there is mail in the mailbox

  THEN open the mailbox and take it out

  there is no action to be taken when the condition is false.

The basic structure of decisions (continued)

- This can be rewritten as:

```
IF there is mail in the mailbox
THEN open the mailbox and take it out
ELSE do nothing
```

Choosing From More Than 2 Options

- There are occasions when we may do one of several different things, depending on some value.
- Imagine that you are going to the bank, where are several different type of transactions. You may wish to:
  - make a deposit
  - cash a check
  - transfer between accounts
  - order new checks or
  - just check your balance.

Choosing From More Than 2 Options (continued)

- We could write this as:

```
IF you want to make a deposit
  THEN process deposit
ELSE IF you want to cash a check
  THEN process check
ELSE IF you want to transfer money
  THEN process transfer
ELSE IF you want to order checks
  THEN process order
ELSE IF you want to check balance
  THEN process balance
  ELSE it must be an error
```

if and ***if-else***

- Some problems may have a set of instructions that are only performed under some conditions. These require an **`if`** construct.
- Other problems may have two or more alternative sets of instructions depending on some condition(s). If there are two alternatives, it requires an if-else construct.

if and ***if-else (continued)***

- The general form is:

```
if expression:
  statement(s)
```

or

```
if expression:
  statement(s)
else:
  statement(s)
```

Python `if` Statement

```
if expression:
  suite
```

- evaluate the boolean (**`True`** or **`False`**)
- if **`True`**, execute all statements in the suite

Warning About Indentation

- Elements of the suite must all be indented the same number of spaces/tabs
- Python only recognizes suites when they are indented the same distance (***<u>standard is 4 spaces</u>***)
- You must be careful to get the indentation right to get suites right.

Python **`if-else`** Statement

```
if boolean expression:
    suite1
else:
    suite2
```

Evaluate the Boolean expression

If **`True`***, run suite1*

If **`False`***, run suite2*

`if`-`else` Example

```
first_int = 10
second_int = 20
if first_int > second_int:
    print("The first int bigger")
else:
    print("The second int is bigger")
```

Simple Boolean Expressions

- Let's take another look at the first part of an **`if`** structure:

  `if (`*condition***`)`**

- A condition is either true or false; it is also called by a more formal name, a ***logical expression*** or ***Boolean expression***, which an combination of terms that is either **true** or **false**.

Relational Operators

- So far, we have seen only one example of this:

  `number < 0`

- We want to know if the relationship is true, i.e., is **`number`** less than 0.
- Since the truth or falsehood depends on this **relationship**, we call < a ***relational operator***.

Relational vs. arithmetic operators

- Compare **`number < 0`** to **`length * width`**.
  - **`number < 0`** produces a true or false
  - **`length * width`** produces a number.
- In both cases, we are combining variables to produce a new result, but the type of result is entirely different.

Relational Operators for Decisions

Operator	Operator's Meaning
`<`	Less than
`>`	Greater than
`<=`	Less than or equal to
`>=`	Greater than or equal to
`==`	Equals
`!=`	Not equal to

Note that `==` *is equality,* `=` *is assignment*

Relational operators (continued)

- Assume that x = 4 y = 6 and z = -2
- Which of the following are true?

```
x+z >= 0
x < z
x <= y
x != y+z
2*x + z == y
```

- The advantage of using variables in these expressions is that the values may change from one program run to another.

Example – Is It Negative?

- Example – Write a program that determine if a number is negative or non-negative
- Our ***algorithm*** (recipe for a program):
  1. Get the number
  2. Print whether its negative or non-negative

Is It Negative? (continued)

1. Get the number
2. Print whether its negative or non-negative

```
2.      IF the number is negative
        2.1     THEN print a message saying that it is negative
        2.2     ELSE print a message saying that it is not negative
```

Is It Negative? (continued)

```
1. Get the number
2. IF the number is negative
   2.1 THEN print a message saying that it is negative
   2.2 ELSE print a message saying that it is not negative
```

```
number = input("Please enter a number?")
number = float(number)
```

Is It Negative? (continued)

```
number = input("Please enter a number?")
number = float(number)
```

```
2. IF the number is negative
   2.1 THEN print a message saying that it is negative
   2.2 ELSE print a message saying that it is not negative
```

```
if number < 0.0:
    print(number, " is a negative number")
else:
    print(number, " is a NOT negative number")
```

IsItNeg.py

```
# Tell a user if a number is negative or
# non-negative

# Ask the user for a number
number = input("Please enter a number?")
number = float(number)

# Print whether the number is negative or
# not
if number < 0.0:
    print(number, " is a negative number")
else:
    print(number, " is a NOT negative number")
```

Example – Calculating Speed

- *Example* - Calculate the speed that you are driving from the distance and time that you have been driving. If you are going over the speed limit, print a warning message.
- We know the following about our problem:

  Available input:
  - Distance in miles
  - Time in hours

  Required output:
  - Speed in miles per hour
  - Warning message (if appropriate)

Example – Calculating Speed (continued)

- We have to perform the following steps:
  1. **Read in the distance in miles and time in hours.**
  2. **Calculate and print the speed.**
  3. **Print the warning if appropriate.**

Example – Calculating Speed (continued)

1. Read in the distance in miles and time in hours.

2. Calculate and print the speed.

3. Print the warning if appropriate.

1.1 Read distance traveled
1.2 Read the time traveled

Example – Calculating Speed (continued)

1.1 Read distance traveled

1.2 Read the time traveled

2. Calculate and print the speed.

3. Print the warning if appropriate.

2.1 Calculate the speed
2.2 Print the speed

Example – Calculating Speed (continued)

1.1 Read distance traveled

1.2 Read the time traveled

2.1 Calculate the speed

2.2 Print the speed

3. Print the warning if appropriate.

3. If the speed > 55
3. 1 then print the warning

Example – Calculating Speed (continued)

1.1 Read distance traveled

1.2 Read the time traveled

2.1 Calculate the speed

2.2 Print the speed

3. If the speed > 55

3. 1 then print the warning

```
miles = input("How many miles have you driven?")
miles = float(miles)
```

Example – Calculating Speed (continued)

```
miles = input("How many miles have you driven?")
miles = float(miles)
```

1.2 Read the time traveled

2.1 Calculate the speed

2.2 Print the speed

3. If the speed > 55

3. 1 then print the warning

```
miles = input("How many miles have you driven?")
miles = float(miles)
```

Example – Calculating Speed (continued)

```
miles = input("How many miles have you driven?")
miles = float(miles)
miles = input("How many miles have you driven?")
miles = float(miles)
```

2.1 Calculate the speed

2.2 Print the speed

3. If the speed > 55

3. 1 then print the warning

```
speed = miles / hours
```

Example – Calculating Speed (continued)

```
miles = input("How many miles have you driven?")
miles = float(miles)
miles = input("How many miles have you driven?")
miles = float(miles)
speed = miles / hours
```

2.2 Print the speed

3. If the speed > 55

3. 1 then print the warning

```
print("You were driving at ", speed, \
           " miles per hour.")
```

Example – Calculating Speed (continued)

```
miles = input("How many miles have you driven?")
miles = float(miles)
miles = input("How many miles have you driven?")
miles = float(miles)
speed = miles / hours
print("You were driving at ", speed, \
           " miles per hour.")
```

3. If the speed > 55

3. 1 then print the warning

```
if speed > 55:
    print("**BE CAREFUL!**",\
             " You are driving too fast!")
```

The Complete **Speed** Program

```
#  Calculate the speed that you are traveling
#  from the distance and time that you have
#  been driving.
#  Print a warning if you are going over the
#  speed limit.

#  Read in the distance in miles and
#  time driven
miles = input("How many miles have you driven?")
miles = float(miles)

hours = input("How many hours did it take?")
hours = float(hours)
```

```
#  Calculate and print the speed
speed = miles / hours
print("You were driving at ", speed, \
           " miles per hour.")

#  Print the warning if appropriate
if speed > 55:
    print("**BE CAREFUL!**",\
            " You are driving too fast!")
```

Error Checking: The **`ConvertPounds`** Program

- Let’s take another look at the program which converts pounds to kilograms.
- Since a weight cannot be a negative number, our program should not accept a negative number as a valid weight.
- Let’s rewrite the program to print an error message if the weight entered in pounds is negative.

Designing **`ConvertPounds2`**

Let's plan our algorithm:

1. Read the weight in pounds
2. If the weight in pounds is negative, print an error message; otherwise calculate and print the weight in kilograms.

Designing **`ConvertPounds2`**(continued)

Let's plan our algorithm:

1. Read the weight in pounds
2. If the weight in pounds is negative, print an error message; otherwise calculate and print the weight in kilograms.

```
2.      IF weight in pounds is negative
        2.1     THEN print an error message
        2.2     ELSE calculate and print the weight
                in kilograms
```

Designing **`ConvertPounds2`**(continued)

Let’s plan our algorithm:

```
1. Read the weight in pounds
2. IF weight in pounds is negative
   2.1  THEN print an error message
   2.2  ELSE calculate and print the weight
        in kilograms
```

```
if lbs < 0:
```

Designing **`ConvertPounds2`**(continued)

Let’s plan our algorithm:

```
1. Read the weight in pounds
if lbs < 0:
   2.1  THEN print an error message
   2.2  ELSE calculate and print the weight
        in kilograms
```

```
print(lbs, " is not a valid weight.")
```

Designing **`ConvertPounds2`**(continued)

Let’s plan our algorithm:

1. Read the weight in pounds

```
if lbs < 0:
    print(lbs, " is not a valid weight.")
else:
    kg = lbs / 2.2;
    print("The weight is ", kg, " kilograms")
```

```
lbs = input("What is the weight in pounds?")
lbs = float(lbs)
```

The Revised **`ConvertPounds`**

```
#   Convert pounds to kilograms
#   Input - weight in pounds
#   Output - weight in kilograms

#   Get the weight in pounds
lbs = input("What is the weight in pounds?")
lbs = float(lbs)

#   Ensure that the weight in pounds is
#   valid. If it is valid, calculate and
#   display the weight in kilograms
if lbs < 0:
    print(lbs, " is not a valid weight.")
else:
    kg = lbs / 2.2;
    print("The weight is ", kg, " kilograms")
```

Constants

- Let's re-examine the statement in our program **`ConvertPounds2`** that does the actual conversion:

```
kg := lbs / 2.2;
```

- Where does come 2.2 from? (There are 2.2 pounds per kilogram)
- How would know why we use 2.2 if we are not familiar with the problem?

Constants (continued)

- We call the constant 2.2 a **literal** because this is ***literally*** the value that we wish to use.
- Any value that we write in a program, whether it is a number or a character or a character string is considered a literal.
- The problem with using literals is that we do not always know why this particular value appears in the program.

Constants (continued)

- Not knowing makes it difficult to understand precisely how the program works and makes it more difficult if we need to correct or modify the program.
- We can write at the beginning of our program:

```
lbsPerKg = 2.2
```

- While this value can be changed within the program, it still allows us to easily identify the significance of the value

Why Use Constants?

- While some values (like inches/ft and lbs/kg) may never change, others will (tax rates, wages, etc.). Now the change only must be made in one place.
- It explains what the value represents.

ConvertPounds3

```
#   Convert pounds to kilograms
#   Input - weight in pounds
#   Output - weight in kilograms

lbsPerKg = 2.2

#   Get the weight in pounds
lbs = input("What is the weight in pounds?")
lbs = float(lbs)

#   Ensure that the weight in pounds is
#   valid. If it is valid, calculate and
#   display the weight in kilograms
if lbs < 0:
    print(lbs, " is not a valid weight.")
else:
    kg = lbs / lbsPerKg;
    print("The weight is ", kg, " kilograms")
```

Compounds Decisions

We saw earlier that not all decisions are simple cases of one or two options depending on a single condition. Consider our choice of transactions at the bank:

```
IF you want to make a deposit
  THEN process deposit
ELSE IF you want to cash a check
  THEN process check
ELSE IF you want to transfer money
  THEN process transfer
ELSE IF you want to order checks
  THEN process order
ELSE IF you want to check balance
  THEN process balance
  ELSE it must be an error
```

Example with Compound Decision

```
score = int(input("Enter a test score>"))

if score >= 90:
    grade = "A"
elif score >= 80:
    grade = "B"
elif score >= 70:
    grade = "C"
elif score >= 60:
    grade = "D"
elif score >= 0:
    grade = "F"

print("Your score was ", score, \
   "which corresonded to "  , grade)
```

Example with Compound Decision

What if:

```
if score <= 90:
    grade = "A"
elif score <= 80:
    grade = "B"
elif score <= 70:
    grade = "C"
elif score <= 60:
    grade = "D"
elif score <= 0:
    grade = "F"

print("Your score was ", score, \
   "which corresonded to "  , grade)
```

Example with Compound Decision

But this will work:

```
if score < 60:
    grade = "F"
elif score < 70:
    grade = "D"
elif score < 80:
    grade = "C"
elif score < 90:
    grade = "B"
elif score < 100:
    grade = "A"

print("Your score was ", score, \
   "which corresonded to " , grade)
```

An Auto Insurance Program

- Example - Write a program to determine the cost of an automobile insurance premium, based on driver's age and the number of accidents that the driver has had.
- The basic insurance charge is $500. There is a surcharge of $100 if the driver is under 25 and an additional surcharge for accidents:

# of accidents	Accident Surcharge
1	50
2	125
3	225
4	375
5	575
6 or more	No insurance

An Auto Insurance Program (continued)

- Available input
  - Number of accidents
  - driver age
- Required output
  - Insurance charge.

Designing the Insurance Program's Algorithm

- Let's start with the basic algorithm:

  1. Input the driver's age and number of accidents.
  2. Determine the insurance charge.
  3. Print the charge and all other relevant information.

Designing the Insurance Program’s Algorithm

1. Input the driver's age and number of accidents.
2. Determine the insurance charge.
3. Print the charge and all other relevant information.

```
age = input("How old is the driver?")
age = int(age)
numAccidents = input("How many accidents has the driver had?")
numAccidents = int(numAccidents)
```

Designing the Insurance Program’s Algorithm

```
age = input("How old is the driver?")
age = int(age)
numAccidents \
     = input("How many accidents has the driver had?")
numAccidents = int(numAccidents)
```

2. Determine the insurance charge.
3. Print the charge and all other relevant information.

```
2.1     IF the driver is under 25
2.1.1       THEN Set the Age Surcharge to $100
2.2     Add on the appropriate surcharge if the driver has had accidents
2.3     Add the surcharges to the rate
```

Designing the Insurance Program’s Algorithm

```
age = input("How old is the driver?")
age = int(age)
numAccidents \
     = input("How many accidents has the driver had?")
numAccidents = int(numAccidents)
```

```
2.1      IF the driver is under 25
2.1.1            THEN Set the Age Surcharge to $100
2.2      Add on the appropriate surcharge if the driver has had accidents
2.3      Add the surcharges to the rate
```

3. Print the charge and all other relevant information.

```
if age < 25:
     ageSurcharge = 100
else:
     ageSurcharge = 0
```

…… … …

```
if age < 25:
      ageSurcharge = 100
Else:
      ageSurcharge = 0
```

2.2 Add on the appropriate surcharge if the driver has had accidents

2.3 Add the surcharges to the rate

3. Print the charge and all other relevant information.

```
if numAccidents == 0:
    accidentSurcharge = 0
elif numAccidents == 1:
    accidentSurcharge = 50
elif numAccidents == 2:
    accidentSurcharge = 125
elif numAccidents == 3:
    accidentSurcharge = 225
elif numAccidents == 4:
    accidentSurcharge = 375
elif numAccidents == 5:
    accidentSurcharge = 575
```

```
…… … …
if age < 25:
       ageSurcharge = 100
else:
       ageSurcharge = 0
if numAccidents == 0:
              accidentSurcharge = 0
else if numAccidents == 1:
              accidentSurcharge = 50
else if numAccidents == 2:
              accidentSurcharge = 125
else if numAccidents == 3:
              accidentSurcharge = 225
else if numAccidents == 4:
       accidentSurcharge = 375
else if numAccidents == 5:
              accidentSurcharge = 575
```

2.3 Add the surcharges to the rate
3. Print the charge and all other relevant information.

```
rate = basicRate + ageSurcharge + accidentSurcharge
print("The total charge is $", rate)
```

The Final Insurance Program

```
#  A program to calculate insurance premiums
#  based on the driver’s age and accident
#  record.
basicRate = 500
ageSurcharge = 0
accidentSurcharge = 0
error = False
tooMany = False

#  Input driver's age and number of
#  accidents
age = input("How old is the driver?")
age = int(age)

numAccidents = input("How many accidents has the
driver had?")
numAccidents = int(numAccidents)
```

```
#  Determine if there is an age surcharge
if age < 0:
    error = True
elif age < 25:
    ageSurcharge = 100
else:
    ageSurcharge = 0
```

```
#  Determine if there is a surcharge
if numAccidents < 0:
    error = True
elif numAccidents == 0:
    accidentSurcharge = 0
elif numAccidents == 1:
    accidentSurcharge = 50
elif numAccidents == 2:
    accidentSurcharge = 125
elif numAccidents == 3:
    accidentSurcharge = 225
elif numAccidents == 4:
    accidentSurcharge = 375
elif numAccidents == 5:
    accidentSurcharge = 575
else:
    tooMany = True
```

```
#  Print the charges
if error:
    print("There has been an error in the", \
                 " data that you supplied")
elif tooMany:
    print("You have had too many accidents", \
                  " for me to insure you.");
else:
    print("The basic rate is $", basicRate)
    if ageSurcharge > 0:
        print("There is an extra ", \
              " surcharge of $", ageSurcharge, \
              " because the driver is ", \
              "under 25.")
    if accidentSurcharge > 0:
        print("There is an extra surcharge", \
             " of $", accidentSurcharge, \
             " because the driver had ", \
             numAccidents, " accident(s).");
```

```
   rate = basicRate + ageSurcharge \
                    + accidentSurcharge
   print("The total charge is $", rate)
```