

# Computer Programming for Non-majors

## 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)

**IF** the light is green ←

→ **THEN** cross the street

→ **ELSE** wait for the light to change

- This decision has three parts:
  - the condition
  - a course of action to follow if the condition is true
  - a course of action to follow if the condition is false.

## 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
```

or

```
if expression :  
    statement
```

```
else :  
    statement
```

## 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

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

## Is It Negative? (continued)

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

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

## IsItNegative.py

```
# Tell a user if a number is negative or  
# non-negative  
  
# Ask the user for a number  
number = float(input("Please enter a 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")
```

## 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 is a 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

<u>Operator</u>	<u>Meaning</u>	<u>Example</u>
==	equals	<b>x == y</b>
!=	is not equal to	<b>1 != 0</b>
>	greater than	<b>x+1 &gt; y</b>
<	less than	<b>x-1 &lt; 2*x</b>
>=	greater than or equal to	<b>x+1 &gt;= 0</b>
<=	less than or equal to	<b>-x +7 &lt;= 10</b>

## Relational operators (continued)

- Assume that  $x = 4$   $y = 6$  and  $z = -2$
- Which of the following are true?
  - $x+z \geq 0$
  - $x < z$
  - $x \leq y$
  - $x \neq y+z$
  - $2*x + z == y$
- The advantage of using variables in these expressions is that the values may change from one program run to another.

## Example – 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

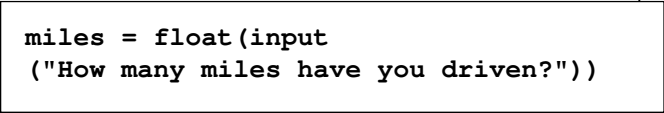
2.1 Calculate the speed

2.2 Print the speed

3. If the speed > 55

3.1 then print the warning

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

A rectangular box containing Python code is positioned below the list of steps. An arrow points from the top-right corner of this box to the text of step 1.1, "1.1 Read distance traveled".

## Example – Calculating Speed (continued)

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

1.2 Read the time traveled

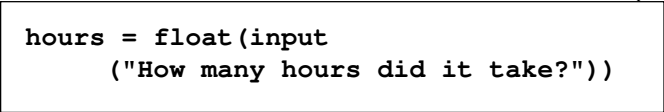
2.1 Calculate the speed

2.2 Print the speed

3. If the speed > 55

3.1 then print the warning

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

A rectangular box containing Python code is positioned below the list of steps. An arrow points from the top-right corner of this box to the text of step 1.2, "1.2 Read the time traveled".

## Example – Calculating Speed (continued)

```
miles = float(input  
    ("How many miles have you driven?"))  
hours = float(input  
    ("How many hours did it take?"))
```


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)

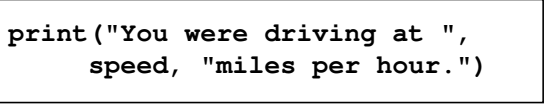
```
miles = float(input  
    ("How many miles have you driven?"))  
hours = float(input  
    ("How many hours did it take?"))  
speed = miles / hours
```

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 = float(input
    ("How many miles have you driven?"))
hours = float(input
    ("How many hours did it take?"))
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 :



## Example – Calculating Speed (continued)

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

if speed > 55 :

3.1 then print the warning

if speed > 55 :



## Example – Calculating Speed (continued)

```
miles = float(input
    ("How many miles have you driven?"))
hours = float(input
    ("How many hours did it take?"))
speed = miles / hours
print("You were driving at ", speed,
    "miles per hour.")
if speed > 55 :
    3.1 then print the warning
```

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

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

# 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

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

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

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

## Designing ConvertPounds2

Let's plan our algorithm:

```
lbs = float(input  
("What is 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

Let's plan our algorithm:

```
lbs = float(input  
    ("What is 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

Let's plan our algorithm:

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

```
if lbs < 0 :
```

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

2.2 ELSE calculate and print the weight  
in kilograms

```
    kg = lbs / 2.2  
    print("The weight is ", kg, " kilograms")
```

## ConvertPounds2.py

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

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

# 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.
- 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.

## Constants (continued)

There is a better way to handle this. While Python does not allow us to declare values as constant (not subject to further revision), we can give these values names and indicate that their values should not and do change within the program.

- Any such constants should appear at the beginning of the program and should have their names in all upper case characters.

## ConvertPounds3.py

```
# Convert pounds to kilograms

# Set the pounds per kilogram at 2.2
POUNDS_PER_KG = 2.2

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

# 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 / POUNDS_PER_KG
    print("The weight is ", kg, " kilograms")
```

## Setting Constants

•The other examples of constants include:

```
WITHHOLDING_RATE = 0.8
PROMPT = 'y'
ANSWER = "yes"
MAX_PEOPLE = 15
INCH_PER_FT = 12
SPEED_LIMIT = 55
```

## 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.

## 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
```



## Compound Decisions (continued)

- Being able to do more than one statement is helpful:

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

## Coding Compound Statements

- Sometimes the decision isn't about choosing one of two options; it's about choosing one of many more options.

## Coding Compound Statements (continued)

- We could write:

```
if color == blue :  
    print "blue"  
    if color == red :  
        print "red"  
        if color == "yellow" :  
            print "yellow"
```

## Coding Compound Statements (continued)

- Instead we can use the reserved word elif to replace else if and avoid the extra indentation

```
if color == blue :  
    print "blue"  
elif color == red :  
    print "red"  
elif color == "yellow" :  
    print "yellow"
```

## 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:

<u># of accidents</u>	<u>Accident Surcharge</u>
1	50
2	125
3	225
4 or more	375

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

## Designing the Insurance Program's Algorithm

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

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

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
```

## Designing the Insurance Program's Algorithm

... ..

```
if age < 25 :  
    ageSurcharge = 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 numAccidents == 0 :  
    accidentSurcharge = 0  
elif numAccidents == 1 :  
    accidentSurcharge = 50  
elif (numAccidents == 2):  
    accidentSurcharge = 125  
elif numAccidents == 3 :  
    accidentSurcharge = 225  
elif numAccidents >= 4 :  
    accidentSurcharge = 375
```

## Designing the Insurance Program's Algorithm

... ..

```
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
```

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)
```

## InsureCar.py

```
# A program to calculate insurance premiums
# based on the driver's age and accident
# record.

basicRate = 500
ageSurcharge = 0
accidentSurcharge = 0

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

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

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

# Determine if there is a surcharge
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
```

```
# Print the charges
print("The basic rate is $", basicRate)
print("The age surcharge is $", ageSurcharge)

print("The accident surcharge is $",
      accidentSurcharge)

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