Introduction to Computer Programming

Decision Structures & Boolean Logic

CSCI-UA.0002
That language is an instrument of human reason, and not merely a medium for the expression of thought, is a truth generally admitted.

-George Boole
Sequence Structures

So far our code has been sequential. Each line of code is executed after the other line of code has executed. What if we wanted multiple things to occur given certain user input?
Sequence Structures

What we have been programming so far is known as a “sequence structure”

Sequence structures are sets of statements that execute in the order in which they appear.

Unfortunately not all programs can be written this way, as there are certain times when we need to deviate from a linear structure and adapt our program based on information provided.
Sequence Structures

1. Start
2. Input a width
3. Input a height
4. Multiply values
5. Output result
6. Finish
Example: Calculating Overtime Pay

If a worker works more than 40 hours in a week he or she is entitled to overtime pay.

Overtime pay is calculated at the rate of 1.5 times the worker’s hourly rate.

This additional rate is only applied to hours worked above the 40 hour limit.
Example: Calculating Overtime Pay

- Input: Hourly rate of pay

- Input: Number of hours worked in 1 week

- Process: If the hours worked is less than 40, simply multiply hourly rate by hours worked

- Process: If the hours worked is greater than 40:
  - Multiply hourly rate by hours worked for 40 hours.
  - Subtract 40 from the total hours to obtain the overtime hours
  - Multiply overtime hours by 1.5 times the rate of pay
  - Add overtime pay to base pay

- Output: Total Pay
Example: Calculating Overtime Pay

The toolset we have developed thus far doesn’t give us the ability to deviate from a linear sequence structure.
The Selection Statement

So we turn to the selection statement which allows us to “ask a question” and respond accordingly.

In its simplest form a selection statement will perform an action only if a certain condition exist.

If that condition is not met, well the action is not performed.
The Selection Statement

In this program we begin by asking a question – “is it cold outside?”

If the answer to this question is yes (aka “True”) then we can execute an alternate set of commands.

Otherwise we can continue with the program as-is.
The Selection Statement

Start

Cold Outside?

Put on a coat

Finish

False

True
The Selection Statement

Start

Gold Outside? True

Put on a coat
Wear a hat
Put on gloves

False

Finish
Selection Statements In Python

- The "if" keyword begins a selection statement.
- "if condition:
  statement
  statement
  statement
  statement"

- Condition to be tested.
- Colon denotes end of condition.
- Statements to be executed if condition is true.
- "Block" of execution must be consistently indented.
Boolean Expressions: Writing Conditions

The trick to writing a selection statement is *in constructing a condition that matches the question you are trying to ask the computer*

All selection statements must have a condition to “test”

Think of conditions as “yes or no” questions. They can only be answered by one of two options – “True” or “False”
Boolean Expressions

if condition:
    statement
    statement
    statement

True or False
Boolean Expressions

Named after George Boole, a 19\textsuperscript{th} century English philosopher and mathematician

Boole developed a system of mathematics that allows us to work with the abstract concepts of “true” and “false”

Boole is considered one of the founders of modern computer science, as his work underpins the way in which modern computers process binary data
Boolean expressions can be used as the condition in an “if” statement.

They are generally formed using “relational operators” which allow you to test to see whether a specific relationship exists between two (or more) values.
Relational Operators

a > b  # is a greater than b ?

a < b  # is a less than b ?

a == b  # is a equal to b ?

a <= b  # is a less than OR
       # equal to b ?

a >= b  # is a greater than OR
       # equal to b ?
ALL Boolean expressions boil down to “True” or “False”

Programmers often say that the expression “evaluates” to “True” or “False”
Writing Boolean Expressions

```python
pen = 10
sword = 7

if pen > sword:
    print ('the pen is mightier than the sword!')
    # pen > sword
    # 10 > 7
    # True
```
Let us Evaluate!

# given these variables

a = 99
b = 7
c = -5
d = 92

# evaluate these expressions

a > b
b < c
b >= c
c <= d
a == b + d
d <= a + c
c != b
Boolean Operator Tips

Don’t confuse `==` with `=`

`=` is used for assigning values to variables
`==` is used for testing to see if two values are identical

Use `!=` if you want to test if two values are different

The `<=` and `>=` operators test for more than one relationship

`<=` tests to see if a value is less than OR equal to another
`>=` tests to see if a value is greater than OR equal to another
Guppies are hardy fish, but they can’t live in all water temperatures.

The acceptable range for guppies is between 72 and 86 degrees Fahrenheit.

Write a program that asks the user for a temperature. Then display one of two messages based on the information provided:

You’re going to freeze your guppy!
You’re going to boil your guppy!
Challenge

Ask the user to guess a number between 1 and 10. Assume they will enter an Integer.

Pick a number between 1 and 10 that is your “secret” number (for example, 5)

If the user types in your secret number, tell them that they win!

If the user types in a number less than or greater than your secret number, tell them that they’re either above or below the number and to try again.
You’re the manager of a large, distributed sales force

You want to create an easy to use tool that will allow your sales staff to do the following:

• Input their monthly sales amount

• Determine if they made their monthly quota of $10,000

• If they made their quota, they are eligible for a bonus of $500

• If they made their quota, they should receive a “Good Job!” message

• At the end of the program you should print out how much their bonus will be ($0 or $500)
Challenge

Start

- Input monthly sales

- Met quota?
  - True
    - Assign bonus of $500
  - False
    - Print "You made your quota!"

Finish
Challenge: Extended

All sales people should receive 1% commission on their sales.

If a sales person made over 50,000, they should receive 5% commission on their sales (instead of 1%) – this is in addition to their $500 bonus for making their quota.

Print out their total take-home amount (bonus + commission) at the end of the program.
Selection Statements in the Wild
Selection Statements in the Wild

How are selection statements used on an MTA metrocard machine?

How many selection statements can you count from your last metrocard purchase?
Simple Selection Statements

The selection statements we have been writing so far have only allowed us to create a single alternate branch of execution.

There are many times when we need to create multiple branches of execution based on the value of a Boolean expression.
The IF - ELSE Structure

The **IF-ELSE** structure allows us to perform one set of statements if a condition is true, and another if it is false.
The IF - ELSE Structure

Start

Input Salary

Salary > 50,000?

False

print "Sorry, you didn't qualify"

True

Qualify user for a loan

print "You qualified!"

Finish
The IF - ELSE Structure

```python
if temperature < 32:
    print ("it’s freezing outside!")

else:
    print ("it’s not so bad outside ...")
```
Challenge

If a worker works more than 40 hours in a week he or she is entitled to overtime pay.

Overtime pay is calculated at the rate of 1.5 times the worker’s hourly rate.

This additional rate is only applied to hours worked above the 40 hour limit.
Challenge

Input: Hourly rate of pay

Input: Number of hours worked in 1 week

Process: If the hours worked is less than 40, simply multiply hourly rate by hours worked

Process: If the hours worked is greater than 40:

Multiply hourly rate by hours worked for 40 hours.

Subtract 40 from the total hours to obtain the overtime hours

Multiply overtime hours by 1.5 times the rate of pay

Add overtime pay to base pay

Output: Total Pay
String Comparison

So far we have been writing Boolean expressions that evaluate based on numeric data.

Example:  \( x > 5; y < 10; z == 100 \)

We can also construct Boolean expressions that can test relationships between strings.

When we compare strings we are essentially reducing them to their zeros and ones and comparing them numerically.
<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
</tr>
<tr>
<td>10</td>
<td>LF</td>
</tr>
<tr>
<td>11</td>
<td>VT</td>
</tr>
<tr>
<td>12</td>
<td>FF</td>
</tr>
<tr>
<td>13</td>
<td>CR</td>
</tr>
<tr>
<td>14</td>
<td>SO</td>
</tr>
<tr>
<td>15</td>
<td>SI</td>
</tr>
<tr>
<td>16</td>
<td>DLE</td>
</tr>
<tr>
<td>17</td>
<td>DC1</td>
</tr>
<tr>
<td>18</td>
<td>DC2</td>
</tr>
<tr>
<td>19</td>
<td>DC3</td>
</tr>
<tr>
<td>20</td>
<td>DC4</td>
</tr>
<tr>
<td>21</td>
<td>NAK</td>
</tr>
<tr>
<td>22</td>
<td>SYN</td>
</tr>
<tr>
<td>23</td>
<td>ETB</td>
</tr>
<tr>
<td>24</td>
<td>CAN</td>
</tr>
<tr>
<td>25</td>
<td>EM</td>
</tr>
<tr>
<td>26</td>
<td>SUB</td>
</tr>
<tr>
<td>27</td>
<td>ESC</td>
</tr>
<tr>
<td>28</td>
<td>FS</td>
</tr>
<tr>
<td>29</td>
<td>GS</td>
</tr>
<tr>
<td>30</td>
<td>RS</td>
</tr>
<tr>
<td>31</td>
<td>US</td>
</tr>
<tr>
<td>32</td>
<td>SP</td>
</tr>
<tr>
<td>48</td>
<td>O</td>
</tr>
<tr>
<td>64</td>
<td>@</td>
</tr>
<tr>
<td>80</td>
<td>P</td>
</tr>
<tr>
<td>96</td>
<td>`</td>
</tr>
<tr>
<td>112</td>
<td>p</td>
</tr>
<tr>
<td>113</td>
<td>q</td>
</tr>
<tr>
<td>114</td>
<td>r</td>
</tr>
<tr>
<td>115</td>
<td>s</td>
</tr>
<tr>
<td>116</td>
<td>t</td>
</tr>
<tr>
<td>101</td>
<td>e</td>
</tr>
<tr>
<td>102</td>
<td>f</td>
</tr>
<tr>
<td>118</td>
<td>v</td>
</tr>
<tr>
<td>103</td>
<td>g</td>
</tr>
<tr>
<td>119</td>
<td>w</td>
</tr>
<tr>
<td>104</td>
<td>h</td>
</tr>
<tr>
<td>120</td>
<td>x</td>
</tr>
<tr>
<td>105</td>
<td>i</td>
</tr>
<tr>
<td>121</td>
<td>y</td>
</tr>
<tr>
<td>106</td>
<td>j</td>
</tr>
<tr>
<td>122</td>
<td>z</td>
</tr>
<tr>
<td>107</td>
<td>k</td>
</tr>
<tr>
<td>123</td>
<td>l</td>
</tr>
<tr>
<td>108</td>
<td>l</td>
</tr>
<tr>
<td>124</td>
<td>m</td>
</tr>
<tr>
<td>109</td>
<td>m</td>
</tr>
<tr>
<td>125</td>
<td>n</td>
</tr>
<tr>
<td>110</td>
<td>n</td>
</tr>
<tr>
<td>126</td>
<td>o</td>
</tr>
<tr>
<td>111</td>
<td>o</td>
</tr>
<tr>
<td>127</td>
<td>DEL</td>
</tr>
</tbody>
</table>
Boolean Operators for Strings

'dog' > 'cat'  # is 'dog' greater than 'cat'?
'fish' < 'alligator'  # is 'fish' less than 'alligator'?
'elephant' == 'tiger'  # are 'elephant' and 'tiger' equivalent?
'bat' != 'honey badger'  # are these strings different?
'bat' > 'back'  # is 'bat' greater than 'back'
Write a program that asks the user for a password.

Check to see if the password that was submitted is equal to the string ‘secret’.

If it is, print out a “welcome” message.

Otherwise, tell them to try again.
Basic String Manipulation

Python has a huge string manipulation library that allows you to interact with and modify strings. We are going to get more in depth with this package later in the semester.

For now we will only be exploring two small functions in this package – lower() and upper()
Basic String Manipulation

The `lower()` function converts the characters in a string to all lowercase, while the `upper()` function converts the characters in a string to all uppercase.

These functions are not built into the Python library directly, but exist inside the “str” module – as such they must be referred to using “dot syntax”
Basic String Manipulation

Example:

```python
string_lc = str.lower('Harry Potter')  # string_lc = 'harry potter'
string_uc = str.upper('Harry Potter')  # string_uc = 'HARRY POTTER'
```
Challenge

Rewrite your password protection program to be case insensitive (i.e. the password “Secret” will also let you into your program)
Challenge

Ask the user to type in two names

Compare the names and print them out in alphabetical order
You can ask Python to count the number of characters contained in a string using the `len()` function.

`len()` returns an integer that represents the total length of a string.

Example:

```python
myname = 'harry'
print (len(myname))  # 5
```
Challenge

Ask the user to input two names

Sort the names in size order and print them out to the user
Sometimes you need to ask “follow up” questions after you’ve evaluated the value of a Boolean expression.

Python allows you to “nest” decision structures inside one another, allowing you to evaluate additional conditions.
Guessing a Number using Nested Decision Structures
Challenge

Re-write the “guess the number” game using a nested decision structure.

If the user guesses the number they win. If they don’t you should tell them to guess higher or lower next time depending on their answer.
Guessing a Number using Nested Decision Structures

```python
secretnumber = 5
usernumber = int(input('Guess a number '))
if usernumber == secretnumber:
    print ('you guessed it!')
else:
    if usernumber < secretnumber:
        print ('your number is too low')
    else:
        print ('your number is too high')
```
Nested Decision Structures

Indentation is key – Python will use the indentation level of a structure to determine its relationship to any previous statements
Challenge

Guppies are hardy fish, but they can’t live in all water temperatures.

The acceptable range for guppies is between 72 and 86 degrees Fahrenheit.

Write a program that asks the user for a temperature. Then display one of three messages based on the information provided:

You’re going to freeze your guppy!

You’re going to boil your guppy!

Your guppy is going to be fine!
Write a program that asks the user to enter in a number greater than or equal to zero and less than or equal to 100. If they do not you should alert them and end the program.

Next, determine the letter grade associated with the number. For example, and A is any grade between 90 and 100. Report the letter grade to the user.
Challenge

You’re working for a small bank that wants to write a program to allow its customers to pre-qualify themselves for a personal loan.

Rules for qualification are as follows:

Borrower must make more than $50,000 per year and be at his or her job for at least 2 years.

The 2 year job requirement can be waived, however, for borrowers making more than $100,000 per year.

Write a program to ask the user for their yearly salary as well as the number of years they have been at their current company. Use the rules above to output the string ‘You qualify’ or ‘You do not qualify’.
Testing a Series of Conditions

Testing a series of conditions using an IF-ELSE structure can result in a large amount of indentations

Sometimes this can cause your code to become difficult to read

Example:

Input: ask the user for a numeric grade (i.e. 95)

Process: convert the grade to its letter format (A through F)

Output: print the letter grade
Grade Determination Program

```python

import math

def determine_grade(g):
    g = float(input('grade '))
    if (g > 90):
        return 'A'
    else:
        if (g > 80):
            return 'B'
        else:
            if (g > 70):
                return 'C'
            else:
                if (g > 60):
                    return 'D'
                else:
                    return 'F'

result = determine_grade(85)
print(result)
```
You can simplify complex IF statements by using the **ELIF** structure.

ELIF is an optional structure that can be placed between your IF and ELSE statements.

It allows you to evaluate additional conditions at the same level as the original IF statement.
If-Elif-Else

```python
# Grade Calculation

g = float(input('grade '))

if g > 90:
    print ('A')
elif g > 80:
    print ('B')
elif g > 70:
    print ('C')
elif g > 60:
    print ('D')
else:
    print ('F')
```
Some notes about using ELIFs:

Conditions are tested in the order in which they are written. Once a condition evaluates to True all future conditions are skipped.

An ELSE statement at the end of a decision structure is considered the “catch all” statement if all conditions above end up failing then the statements inside the ELSE block will execute.
However, using an ELSE statement at the end of your decision structure is optional.

There is no logical need for an IF-ELIF-ELSE statement. You can always write a program without it by using a standard IF-ELSE block.

The advantage of an IF-ELIF-ELSE statement is that your code may end up being more readable / understandable.
All programming languages provide a set of "logical operators"

These operators can be used to create complex Boolean expressions that evaluate more than one condition at the same time
Logical Operators

```python
x = 10
y = 5
a = 20
b = 25

if x > y and a < b:
    print ('yes!')
else:
    print ('no!')
```
Logical Operators

Logical operators are used to combine Boolean expressions into a composite Boolean expression.

There are three main logical operators that we use regularly in programming:

- and
- or
- not
The ‘and’ Operator

“and” can be used to combine two Boolean expressions.

The resulting Boolean expression will evaluate to be True if the two Boolean expressions it is connecting both evaluate to be True.

| True   | and | True   | => | True   |
| True   | and | False  | => | False  |
| False  | and | True   | => | False  |
| False  | and | False  | => | False  |
The ‘and’ Operator

```python
a = 5
b = 10
print (a > b and a > 1)
print (a > 1 and b > a)
print (a == 5 and b < 100)
print (a > 1 and b < 1 and b > a)
print (a > 1 and b > 1 and b > a)
```
The ‘and’ Operator

```python
salary = float(input('How much do you make? '))
years = float(input('How long have you been at your job? '))

if salary >= 50000 and years >= 2:
    print ('You qualify for a loan!')
else:
    print ('You do not qualify for a loan')
```
The ‘or’ Operator

“or” can also be used to combine two Boolean expressions

The resulting Boolean expression will evaluate to be True if EITHER of Boolean expressions it is connecting evaluates to be True

True  or True  => True
True  or False => True
False or True  => True
False or False => False
The ‘or’ Operator

a = 5
b = 10

print (a > b or a > 1)
print (a > 1 or b > a)
print (a == 5 or b < 100)
print (a > 1 or b < 1 or b > a)
print (a > 1 or b > 1 or b > a)
temp = float(input('What is the temperature of your fish tank? '))

if temp < 72 or temp > 86:
    print("The temperature is too extreme!")
The ‘not’ Operator

The “not” operator is a unary operator that reverses the logical value of its argument. This means that it will “flip” a True value into a False value, and vice versa.

*unary, in mathematics, is typically an operation with only one operand*
The ‘not’ Operator

```python
username = input('username? ')

if not (username == 'Harry'):
    print("invalid input!")

else:
    print("Welcome, Harry!")
```
Challenge

Write a program that asks a user for a username and a password.

Check to see if BOTH the username and password are correct.

If so, provide a Welcome message to the user.

If not, provide a Login Failure message to the user.
next steps:

begin “Self Paced Learning Module # 4”

work on Assignment #3: ‘Control Structures’