TEC222 Introduction to Programming with Python

Lecture 4

Decision Structures

From last week….

LISTS

The list Object

- A list is an ordered sequence of Python objects
  - Objects can be of any type
  - Objects do not have to all be the same type.
  - Constructed by writing items enclosed in square brackets … items separated by commas.

```
team = ["Seahawks", 2014, "CenturyLink Field"]
nums = [5, 10, 4, 5]
words = ["spam", "ni"]
```

The list Object

- List operations (The lists team, num, and words given in previous slide)

Function or Method	Example	Value	Description
len	len(words)	2	number of items in list
max	max(numbers)	10	greatest (items must have same type)
min	min(numbers)	4	least (items must have same type)
sum	sum(nums)	42	total (items must be numbers)
count	nums.count(5)	2	number of occurrences of an object
index	nums.index(4)	2	index of first occurrence of an object
reverse	words.reverse()	["ni", "spam"]	reverses the order of the items

The list Object

- List operations (The lists team, num, and words given in previous slide)

reverse	words.reverse()	["ni", "spam"]	reverses the order of the items
clear	team.clear()	[]	[] is the empty list
append	nums.append(7)	[5, 10, 4, 5, 7]	inserts object at end of list
extend	nums.extend([1, 2])	[5, 10, 4, 5, 1, 2]	inserts new list's items at end of list
del	del team[−1]	["Seahawks", 2014]	removes item with stated index
remove	nums.remove(5)	[10, 4, 5]	removes first occurrence of an object
insert	nums.insert(1, "wink")	["spam", "wink", "ni"]	insert new item before item of given index
+	['a', 1] + [2, 'b']	['a', 1, 2, 'b']	concatenation; same as ['a', 1].extend([2,'b'])
*	[0] * 3	[0, 0, 0]	list repetition

The list Object

- Example: Program requests five grades as input, displays average after dropping two lowest grades

```
## Calculate average of grades
grades = []   # Create the variable grades and assign it the empty list.
num = float(input("Enter the first grade: "))
grades.append(num)
num = float(input("Enter the second grade: "))
grades.append(num)
num = float(input("Enter the third grade: "))
grades.append(num)
num = float(input("Enter the fourth grade: "))
grades.append(num)
num = float(input("Enter the fifth grade: "))
grades.append(num)
```

The list Object

- Program requests five grades as input, displays average after dropping two lowest grades

```
grades.append(num)
minimumGrade = min(grades)
grades.remove(minimumGrade)
minimumGrade = min(grades)
grades.remove(minimumGrade)
average = sum(grades) / len(grades)
print("Average Grade: {0:.2f}".format(average))
```

[Run]

```
Enter the first grade: 89
Enter the second grade: 77
Enter the third grade: 82
Enter the fourth grade: 95
Enter the fifth grade: 81
Average Grade: 88.67
```

Slices

- A slice of a list is a sublist specified with colon notation
  - Analogous to a slice of a string
- Meanings of slice notations

Slice Notation	Meaning
list1[*m*:*n*]	list consisting of the items of *list1* having indices *m* through *n* – 1
list1[:]	a new list containing the same items as *list1*
list1[*m*:]	list consisting of the items of *list1* from list1[*m*] through the end of *list1*
list1[:*m*]	list consisting of the items of *list1* from the beginning of *list1* to the element having index *m* – 1

Slices

- Examples of slices where list1 = ['a', 'b', 'c', 'd', 'e', 'f'].

Example	Value
list1[1:3]	['b', 'c']
list1[-4:-2]	['c', 'd']
list1[:4]	['a', 'b', 'c', 'd'']
list1[4:]	['e', 'f']
list1[:]	['a', 'b', 'c', 'd', 'e', 'f']
del list1[1:3]	['a', 'd', 'e', 'f']
list1[2:len(list1)]	['c', 'd', 'e', 'f']
(list1[1:3])[1]	'c' (This expression is usually written as list1[1:3][1])
list1[3:2]	[], the list having no items; that is, the empty list

The split and join Methods

- Split method turns single string into list of substrings
- Join method turns a list of strings into a single string.
- Notice that these methods are inverses of each other

The split and join Methods

- These statements each display list ['a', 'b', 'c'].

```
print("a,b,c".split(','))
print("a**b**c".split('**'))
print("a\nb\nc".split())
print("a b c".split())
```

The split and join Methods

- Program shows how join method used to display items from list of strings.

```
line = ["To", "be", "or", "not", "to", "be."]
print(" ".join(line))
krispies = ["Snap", "Crackle", "Pop"]
print(", ".join(krispies))
```

[Run]

```
To be or not to be.
Snap, Crackle, Pop
```

Text Files

- Text file is a simple file consisting of lines of text with no formatting
  - Text file can be created with any word processor
  - Python program can access the values

Text Files

- Given

```
infile = open("Data.txt", 'r')
listName = [line.rstrip() for line in infile]
infile.close()
```

  - Program opens for reading a file of text
  - Strips newline characters
  - Characters placed into a list

Text Files

- Furthermore

```
infile = open("Data.txt", 'r')
listName = [eval(line) for line in infile]
infile.close()
```

  - Suppose data is all numbers,
  - Previous code produces list of strings, each a number
  - Place the numbers into a list with this code

The tuple Object

- Tuples, like lists, are ordered sequences of items
- Difference – tuples cannot be modified in place
  - Have no append, extend, or insert method
- Items of tuple cannot be directly deleted, sorted, or altered

The tuple Object

- All other list functions and methods apply
  - Items can be accessed by indices
  - Tuples can be sliced, concatenated, and repeated
- Tuples written as comma-separated sequences enclosed in parentheses
  - Can also be written without the parentheses.

The tuple Object

- Program shows tuples have several of same functions as lists.

```
t = 5, 7, 6, 2
print(t)
print(len(t), max(t), min(t), sum(t))
print(t[0], t[-1], t[:2])
```

[Run]

```
(5, 7, 6, 2)
4 7 2 20
5 2 (5, 7)
```

The tuple Object

- Program swaps values of two variables

```
x = 5
y = 6
x, y = y, x
print(x, y)
```

[Run]

```
6 5
```

Nested Lists

- Beside numbers or strings, items can be lists or tuples.
- Consider a list of tuples named L
  L[0] is the first tuple
  L[0][0] is the first item in the first tuple
- And L[-1] is the last tuple
  L[-1][-1] is the last item in the last tuple

Nested Lists

- Program manipulates
  regions contains four tuples, each tuple gives name and 2010 population (in millions) of a region

```
regions = [("Northeast", 55.3), ("Midwest", 66.9),
           ("South", 114.6), ("West", 71.9)]
print("The 2010 population of the", regions[1][0], "was", regions[1][1],
      "million.")
totalPop = regions[0][1] + regions[1][1] + regions[2][1] + regions[3][1]
print("Total 2010 population of the U.S: {0:.1f} million.".format(totalPop))
```

[Run]

```
The 2010 population of the Midwest was 66.9 million.
Total 2010 population of the U.S: 308.7 million.
```

Immutable and Mutable Objects

- An object is an entity
  - Holds data.
  - Has operations and/or methods that can manipulate the data.
- When variable created with assignment statement
  - Value on the right side becomes an object in memory
  - Variable references (points to) object

Immutable and Mutable Objects

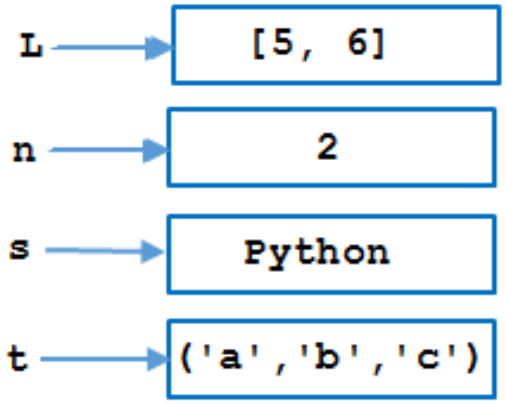
- When list altered
  - Changes made to the object in list's memory location
- Contrast when value of variable is number, string, or tuple … when value changed,
  - Python designates a new memory location to hold the new value
  - And the variable references that new object

Immutable and Mutable Objects

- Another way to say this
  - Lists can be changed in place
  - Numbers, strings, and tuples cannot
- Objects changed in place are mutable
- Objects that cannot be changed in place are immutable

Immutable and Mutable Objects

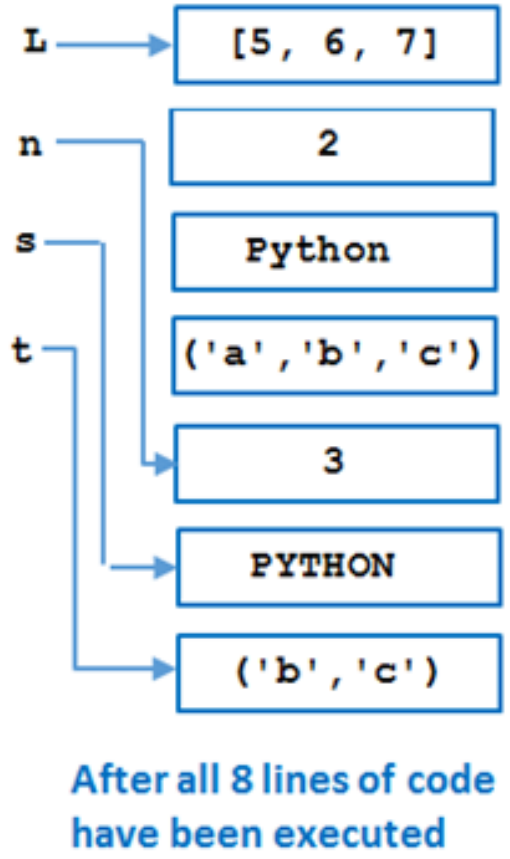
```
L = [5, 6]
n = 2
s = "Python"
t = ('a','b','c')
L.append(7)
n += 1
s = s.upper()
t = t[1:]
```



Memory allocation corresponding to a program.

Immutable and Mutable Objects

```
L = [5, 6]
n = 2
s = "Python"
t = ('a','b','c')
L.append(7)
n += 1
s = s.upper()
t = t[1:]
```



Memory allocation corresponding to a program.

Copying Lists

- Consider results of this program

```
list1 = ['a', 'b']  # Lists are mutable objects.
list2 = list1       # list2 will point to the same memory location as list1
list2[1] = 'c'      # Changes the value of the second item in the list object
print(list1)
```

[Run]

```
['a', 'c']
```

- All because lists are mutable

Copying Lists

- Now note change in line 2

```
list1 = ['a', 'b']  # Lists are mutable objects.
list2 = list(list1) # list2 now points to different memory location
list2[1] = 'c'      # Changes the value of the second item in the list object
print(list1)
```

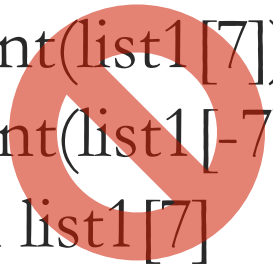
[Run]

```
['a', 'b']
```

- Third line of code will not affect memory location pointed to by list1

Indexing, Deleting, and Slicing Out of Bounds

- Python does not allow out of bounds indexing for individual items in lists and tuples
  - But does allow it for slices
- Given list1 = [1, 2, 3, 4, 5]
  - Then
    ```
    print(list1[7])
    print(list1[-7])
    del list1[7]
    ```



Indexing, Deleting, and Slicing Out of Bounds

- If left index in slice too far negative
  - Slice will start at the beginning of the list
- If right index is too large,
  - Slice will go to the end of the list.

```
list1[-10:10] is [1, 2, 3, 4, 5]
list1[-10:3] is [1, 2, 3]
list1[3:10] is [4, 5]
del list1[3:7] is [1, 2, 3]
```

Determine the output

```
nations = "France\nEngland\nSpain\n"
countries = nations.split()
print(countries)
```

```
# The three lines of Abc.txt contain a b, c, d
infile = open("Abc.txt", 'r')
alpha = [line.rstrip() for line in infile]
infile.close()
word = ("").join(alpha)
print(word)
```

Determine the output

```
ships = ["Nina", "Pinta", "Santa Maria"]
print(ships[-5:2])
```

```
answer = ["Yes!", "No!", "Yes!", "No!", "Maybe."]
num = answer.index("No!")
print(num)
```

Exercise

- Write a program that requests a two-part name and then displays the name in the form "lastName, firstName".

```
Enter a 2-part name: John Doe
Revised form: Doe, John
```

STRUCTURES THAT CONTROL FLOW

Relational and Logical Operators

- A condition is an expression
  - Involving relational operators (such as < and >=)
  - Logical operators (such as and, or, and not)
  - Evaluates to either True or False
- Conditions used to make decisions
  - Control loops
  - Choose between options

ASCII Values

- ASCII values determine order used to compare strings with relational operators.
- Associated with keyboard letters, characters, numerals
  - ASCII values are numbers ranging from 32 to 126.
- A few ASCII values.

32 (space)	48 0	66 B	122 z
33 !	49 1	90 Z	123 {
34 “	57 9	97 a	125 }
35 #	65 A	98 b	126 ~

ASCII Values

- The ASCII standard also assigns characters to some numbers above 126.
- A few of the higher ASCII values.

162 ¢	177 ±	181 µ	190 ¾
169 ©	178 ²	188 ¼	247 ÷
176 °	179 ³	189 ½	248 ø

- Functions chr(n) and ord(str) access ASCII values

Relational Operators

- Relational operator less than (<) can be applied to
  - Numbers
  - Strings
  - Other objects
- For strings, the ASCII table determines order of characters

Relational Operators

Python Notation	Numeric Meaning	String Meaning
==	equal to	identical to
!=	not equal to	different from
<	less than	precedes lexicographically
>	greater than	follows lexicographically
<=	less than or equal to	precedes lexicographically or is identical to
>=	greater than or equal to	follows lexicographically or is identical to
in		substring of
not in		not a substring of

Relational operators.

Relational Operators

- Some rules
  - An int can be compared to a float.
  - Otherwise, values of different types cannot be compared
  - Relational operators can be applied to lists or tuples

```
[3, 5] < [3, 7]
[3, 5] < [3, 5, 6]
[3, 5, 7] < [3, 7, 2]
[7, "three", 5] < [7, "two", 2]
```

Relational Operators

- Determine whether following conditions evaluate to True or False

(a) 1 <= 1
(b) 1 < 1
(c) "car" < "cat"
(d) "Dog" < "dog"
(e) "fun" in "refunded"

Relational Operators

- Determine whether following conditions evaluate to True or False
- Suppose the variables a and b have values 4 and 3, and the variables c and d have values “hello” and “bye”.

(a) (a + b) < (2 * a)

(b) (len(c) − b) == (a/2)

(c) c < (“good” + d)

Sorting the Items in a List

- Items in a list of items having same data type can be ordered with the sort method
- Program illustrates how Python orders two simple lists

```
list1 = [6, 4, -5, 3.5]
list1.sort()
print(list1)
list2 = ["ha", "hi", 'B', '7']
list2.sort()
print(list2)
```

[Run]

```
[-5, 3.5, 4, 6]
['7', 'B', 'ha', 'hi']
```

Sorting the Items in a List

- Items in a complicated list of strings.

```
list1 = [chr(177), "cat", "car", "Dog", "dog", "8-ball", "5" + chr(162)]
list1.sort()
print(list1)
```

[Run]

```
['5¢', '8-ball', 'Dog', 'car', 'cat', 'dog', '±']
```

- Items in a list of tuples

```
monarchs = [("George", 5), ("Elizabeth", 2), ("George", 6), ("Elizabeth", 1)]
monarchs.sort()
print(monarchs)
```

[Run]

```
[('Elizabeth', 1), ('Elizabeth', 2), ('George', 5), ('George', 6)]
```

Logical Operators

- Enables combining multiple relational operators
- Logical operators are the reserved words and, or, and not
- Conditions that use these operators are called compound conditions

Logical Operators

- Given: cond1 and cond2 are conditions
  - cond1 and cond2 true only if both conditions are true
  - cond1 or cond2 true if either or both conditions are true
  - not cond1 is false if the condition is true, true if the condition is false

Logical Operators

- Given n = 4, answ = "Y" Determine expressions = True or False

(a) (2 < n) and (n < 6)
(b) (2 < n) or (n == 6)
(c) not (n < 6)
(d) (answ == "Y") or (answ == "y")
(e) (answ == "Y") and (answ == "y")
(f) not (answ == "y")
(g) ((2 < n) and (n == 5 + 1)) or (answ == "No")
(h) ((n == 2) and (n == 7)) or (answ == "Y")
(i) (n == 2) and ((n == 7) or (answ == "Y"))

Short-Circuit Evaluation

- Consider the condition cond1 and cond2
  - If Python evaluates cond1 as false, it does not bother to check cond2
- Similarly with cond1 or cond2
  - If Python finds cond1 true, it does not bother to check further
- Think why this feature helps for
  (number != 0) and (m == (n / number))

The bool Data Type

- Objects True and False are said to have Boolean data type
  - Of data type bool.
- What do these lines display?

```
x = 2
y = 3
var = x < y
print(var)
```

```
x = 5
print((3 + x) < 7)
```

Methods that Return Boolean Values

- Given: strings str1 and str2
  - str1.startswith(str2)
  - str1.endswith(str2)
- For determining the type of an item
  - isinstance(item, dataType)

Methods that Return Boolean Values

- Methods that return either True or False.

Method	Returns True when
str1.isdigit()	all of *str1*'s characters are digits
str1.isalpha()	all of *str1*'s characters are letters of the alphabet
str1.isalnum()	all of *str1*'s characters are letters of the alphabet or digits
str1.islower()	*str1* has at least 1 alphabetic character and all of its alphabetic characters are lowercase
str1.isupper()	*str1* has at least 1 alphabetic character and all of its alphabetic characters are uppercase
str1.isspace()	*str1* contains only whitespace characters

Methods that return either True or False.

Simplifying Conditions

- Lists or tuples can sometimes be used to simplify long compound conditions

```
(state == "MD") or (state == "VA") or (state == "WV")  or (state == "DE")
```

can be replaced with the condition

```
state in ["MD", "VA", "WV", "DE"]
```

```
(x > 10) and (x <= 20)
```

can be replaced with the condition

```
10 < x <= 20
```

```
(x <= 10) or (x > 20)
```

can be replaced with the condition

```
not(10 < x <= 20)
```

Questions ?

if-else Statements

- Also known as branching structures
- Allow program to decide on course of action
  - Based on whether a certain condition is true or false.
- Form:

```
if condition:
    indented block of statements
else:
    indented block of statements
```

if-else Statements

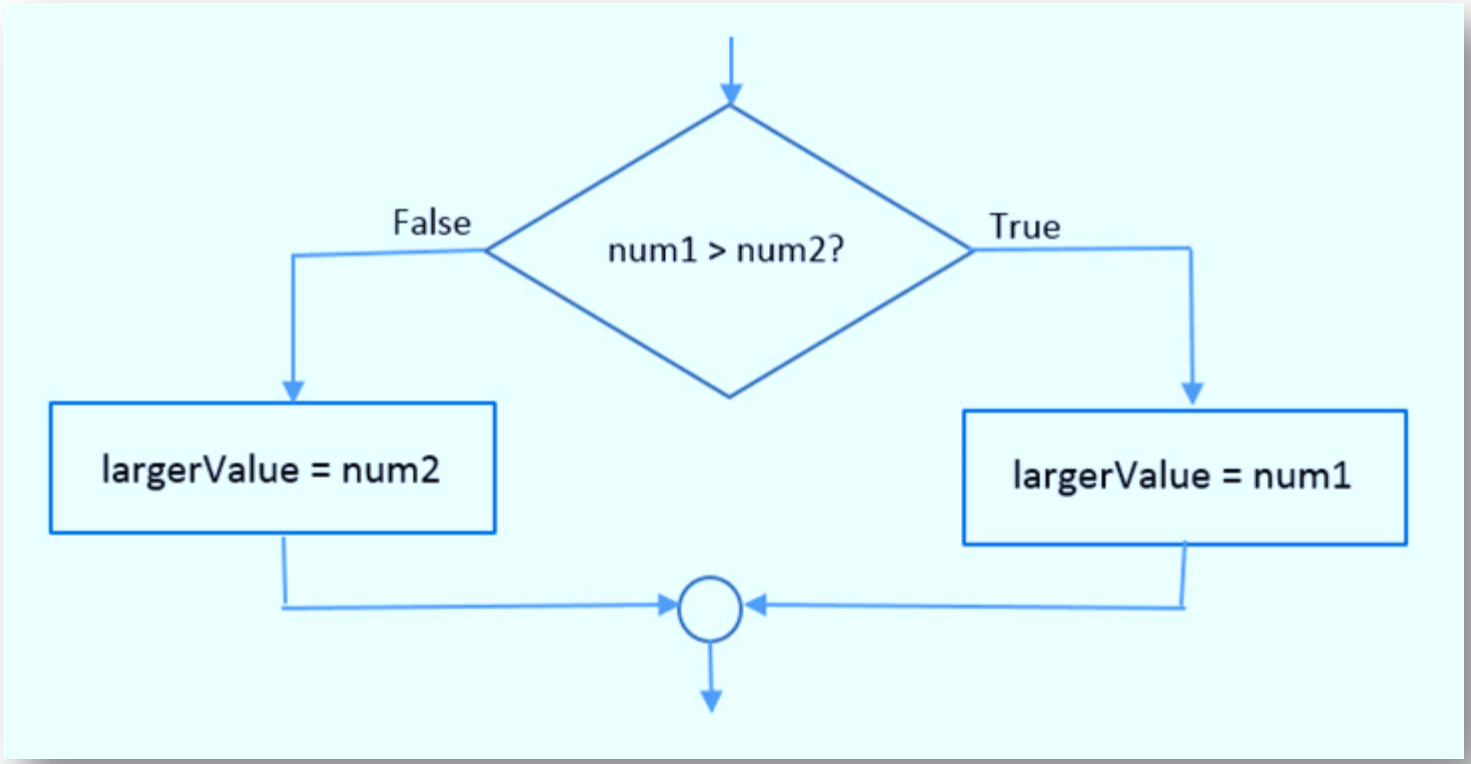
- Program finds larger of two numbers input by user.

```
## Determine the larger of two numbers.
# Obtain the two numbers from the user.
num1 = eval(input("Enter the first number: "))
num2 = eval(input("Enter the second number: "))
# Determine and display the larger value.
if num1 > num2:
    largerValue = num1  # execute this statement if the condition is true
else:
    largerValue = num2  # execute this statement if the condition is false
print("The larger value is", str(largerValue) + ".")
```

[Run]

```
Enter the first number: 3
Enter the second number: 7
The larger value is 7.
```

if-else Statements



Flowchart for the if-else statement in Example 1.

if-else Statements

- if-else statement in program has relational operators in its condition.

```
## A quiz.
# Obtain answer to question.
answer = eval(input("How many gallons does a ten-gallon hat hold? "))
# Evaluate answer.
if (0.5 <= answer <= 1):
    print("Good, ", end="")
else:
    print("No, ", end="")
print("it holds about 3/4 of a gallon.")
```

[Run]

```
How many gallons does a ten-gallon hat hold? 10
No, it holds about 3/4 of a gallon.
```

if Statements

- The else part of an if-else statement can be omitted
- When the condition is false
  - Execution continues with line after the if statement block

if Statements

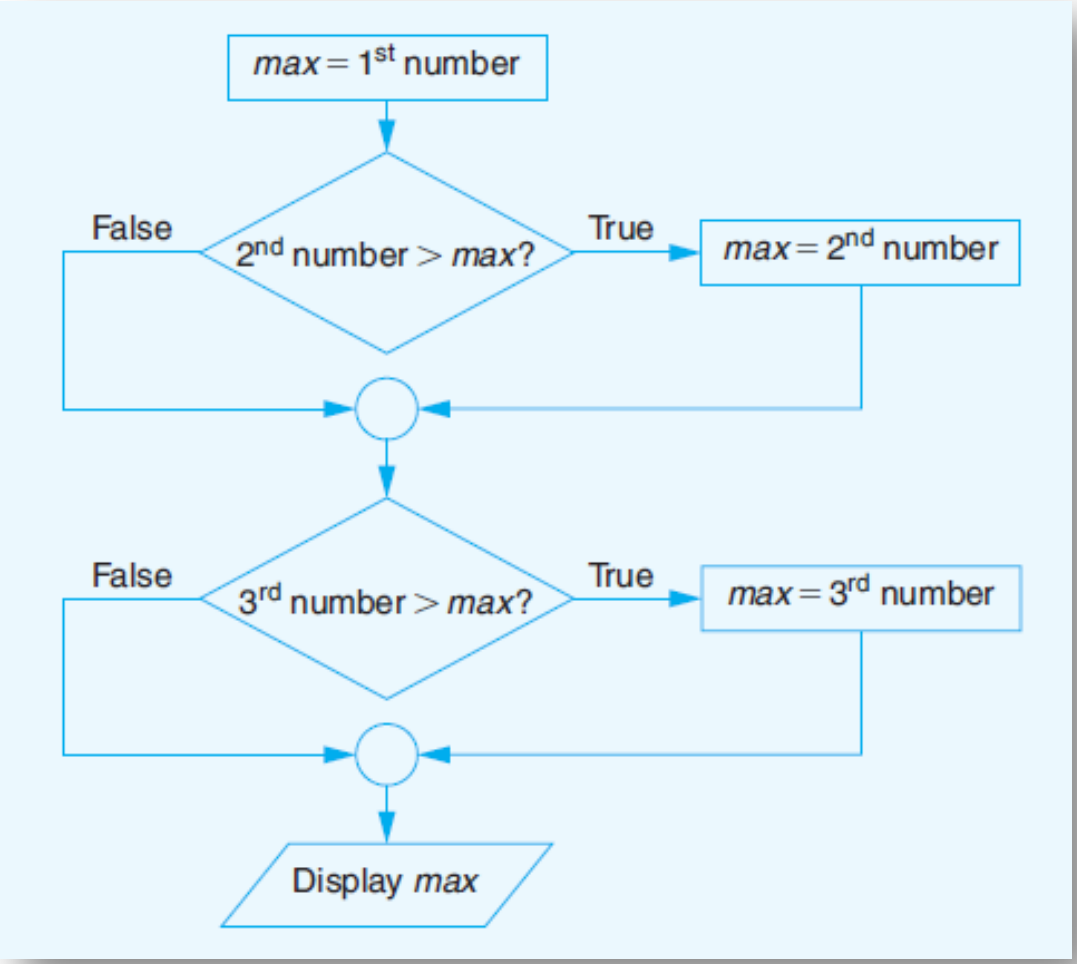
- Program contains two if statements

```
## Find the largest of three numbers.
# Input the three numbers.
firstNumber = eval(input("Enter first number: "))
secondNumber = eval(input("Enter second number: "))
thirdNumber = eval(input("Enter third number: "))
# Determine and display the largest value.
max = firstNumber
if secondNumber > max:
    max = secondNumber
if thirdNumber > max:
    max = thirdNumber
print("The largest number is", str(max) + ".")
```

[Run]

```
Enter first number: 3
Enter second number: 7
Enter third number: 4
The largest number is 7.
```

if Statements



Flowchart for Example 3.

Nested if-else Statements

- Indented blocks of if-else and if statements can contain other if-else and if statements
  - The if-else statements are said to be nested
- Consider the task of interpreting a beacon
  - The color of the beacon light atop Boston's old John Hancock building forecasts the weather

Steady blue, clear view.
Flashing blue, clouds due.
Steady red, rain ahead.
Flashing red, snow instead.

Nested if-else Statements

- Program requests a color (Blue or Red) and a mode (Steady or Flashing) as input
  - Then displays the weather forecast

```
## Interpret weather beacon.
# Obtain color and mode.
color = input("Enter a color (BLUE or RED): ")
mode = input("Enter a mode (STEADY or FLASHING): ")
color = color.upper()
mode = mode.upper()
# Analyze responses and display weather forecast.
result = ""
if color == "BLUE":
    if mode == "STEADY":
        result = "Clear View."
    else:    # mode is FLASHING
        result = "Clouds Due."
```

Nested if-else Statements

- Program requests a color (Blue or Red) and a mode (Steady or Flashing) as input
  - Then displays the weather forecast

```
        else:    # mode is FLASHING
            result = "Clouds Due."
    else:     # color is RED
        if mode == "STEADY":
            result = "Rain Ahead."
        else:    # mode is FLASHING
            result = "Snow Ahead."
    print("The weather forecast is", result)
```

[Run]

```
Enter the color (blue or red): RED
Enter the mode (steady or flashing): STEADY
The weather forecast is Rain Ahead.
```

Nested if-else Statements

- Program requests costs, revenue for company
  - Displays "Break even", profit, or loss

```
## Evaluate profit.
# Obtain input from user.
costs = eval(input("Enter total costs: "))
revenue = eval(input("Enter total revenue: "))
# Determine and display profit or loss.
if costs == revenue:
    result = "Break even."
else:
    if costs < revenue:
        profit = revenue - costs
        result = "Profit is ${0:,.2f}.".format(profit)
```

Nested if-else Statements

- Program requests costs, revenue for company
  - Displays "Break even", profit, or loss

```
            profit = revenue - costs
            result = "Profit is ${0:,.2f}.".format(profit)
        else:
            loss = costs - revenue
            result = "Loss is ${0:,.2f}.".format(loss)
print(result)
```

[Run]

```
Enter total costs: 9500
Enter total revenue: 8000
Loss is $1,500.00.
```

The elif Clause

- Allows for more than two possible alternatives with inclusion of elif clauses.

```
if condition1:
    indented block of statements to execute if condition1 is true
elif condition2:
    indented block of statements to execute if condition2 is true
    AND condition1 is not true
elif condition3:
    indented block of statements to execute if condition3 is true
    AND both previous conditions are not true
else:
    indented block of statements to execute if none of the above
    conditions are true
```

The elif Clause

- Program reports if the two numbers are equal.

```
## Determine the larger of two numbers.
# Obtain the two numbers from the user.
num1 = eval(input("Enter the first number: "))
num2 = eval(input("Enter the second number: "))
# Determine and display the larger value.
if num1 > num2:
    print("The larger value is", str(num1) + ".")
elif num2 > num1:
    print("The larger value is", str(num2) + ".")
else:
    print("The two values are equal.")
```

[Run]

```
Enter the first number: 7
Enter the second number: 7
The two values are equal.
```

Input Validation with if-elif-else Statements

- Program uses the method isdigit to guard against improper input.

```
## Request two numbers and find their sum. Validate the input.
num1 = input("Enter first number: ")
num2 = input("Enter second number: ")
# Display sum if entries are valid. Otherwise, inform
# the user where invalid entries were made.
if num1.isdigit() and num2.isdigit():
    print("The sum is", str(eval(num1) + eval(num2)) + ".")
elif not num1.isdigit():
    if not num2.isdigit():
        print("Neither entry was a proper number.")
    else:
        print("The first entry was not a proper number.")
```

Input Validation with if-elif-else Statements

- Program uses the method isdigit to guard against improper input.

```
        print("Neither entry was a proper number.")
    else:
        print("The first entry was not a proper number.")
else:
    print("The second entry was not a proper number.")
```

[Run]

```
Enter first number: 5
Enter second number: six
The second entry was not a proper number.
```

True and False

- Program illustrates truth values of objects

```
## Illustrate Boolean values.
if 7:
    print("A nonzero number is true.")
else:
    print("The number zero is false.")
if []:
    print("A nonempty list is true.")
else:
    print("An empty list is false.")
if ["spam"]:
    print("A nonempty list is true.")
else:
    print("The empty list is false.")
```

[Run]

```
A nonzero number is true.
An empty list is false.
A nonempty list is true.
```