Computer Programming I

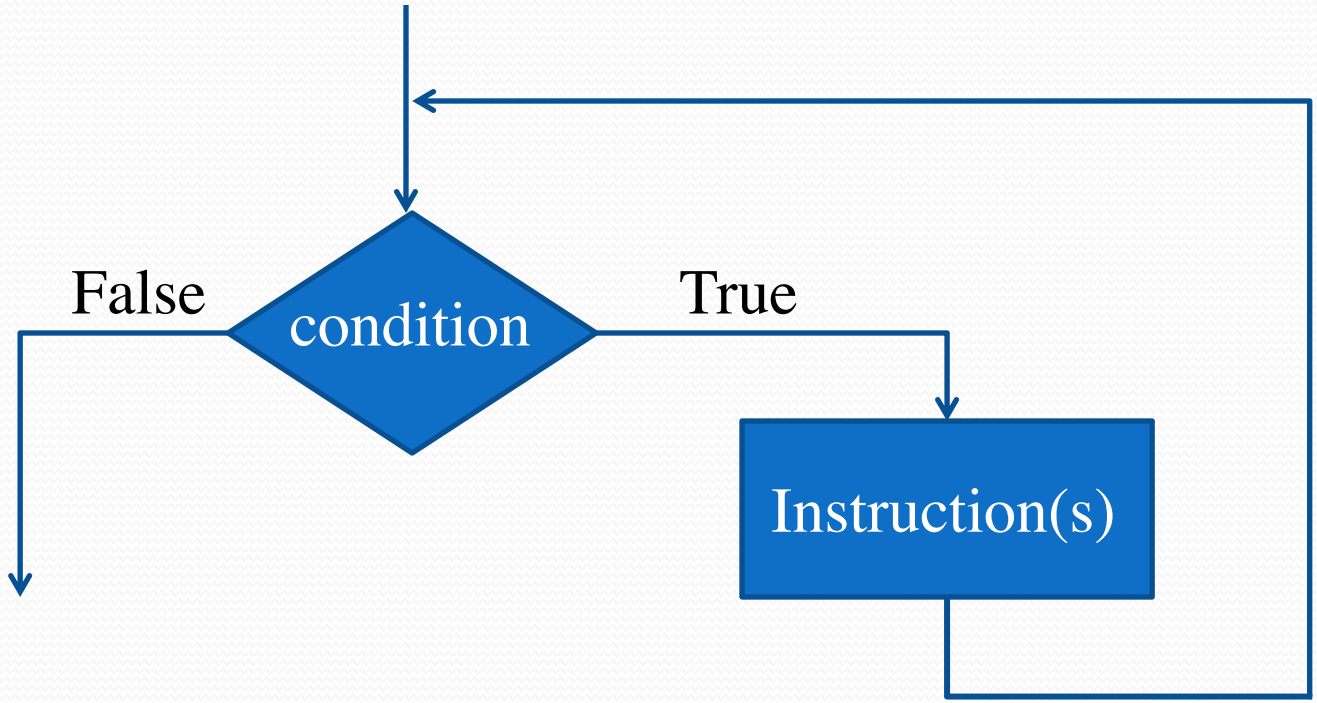
Loops

Introduction to Repetition

- Programs often need to repeat a specific task
  - Disadvantages of duplicated code
    - Makes the program large
    - Time consuming
    - Any changes to one part must be repeated in all parts
    - Sometimes you don't know how many times something needs to be repeated.
- Repetition structures (*loops*) provide a better way
  - Write code once and computer repeats it as many times as needed

The `while` loop

A `while` loop executes a block of code as long as the loop's condition is `True`



The `while` Loop

- The **`while`** loop gets its name from the way it works:
  - *While a condition is true, perform some task*
  - *As long as an expression is true, perform some task*
- It has two parts:
  1. Boolean expression that is tested for **`True/False`**
  2. One or more statements that are repeated as long as the Boolean expression is true → Loop Body
- Each time the loop executes its statement(s) is called an *iteration* of the loop

The `while` loop

- Syntax:

```
while condition :
    Instruction(s)
```

- Example:

```
stars = 1
while stars <= 10 :
    print ('*', end = ' ')
    stars = stars + 1
```

Output:

```
* * * * * * * * * * *
```

Terminology

- Loop Expression: The Boolean expression deciding if the loop should continue (A.K.A. condition)
- Loop Body: Indented code executed repeatedly
- Sentinel Value: A value that is tested against to decide if the loop should continue
- Loop Variable: A variable that counts the number of iterations
- Accumulator: A variable used to collect a total during a loop

The `while` loop - Example

- Identify the loop expression, sentinel value, loop body, loop variable and accumulator:

```
userValue = 0
total = 0
while userValue >= 0 :
    total = total + userValue
    userValue = int(input('Enter a positive value: '))
print('sum of values = ', total)
```

Sample run:

```
Enter a positive value: 9
Enter a positive value: 10
Enter a positive value: 7
Enter a positive value: -1
sum of values =  26
```

Counting Shortcuts

- Remember the shortcut assignment operators?
  - Very common to use them in loops
- The += operator adds and changes a value.

```
a += 1 #is the same as
a = a + 1
```

- Most common operators are supported in this manner.
  - `a -= 2`
  - `a *= 9`
  - `a /= 2`

Counting loop example

```
counter = 0
while counter < 10 :
   print("Counter has value", counter)
   counter += 1   #counter = counter + 1
print("After Loop", counter)
```

Sample run :

```
Counter has value 0
Counter has value 1
Counter has value 2
Counter has value 3
Counter has value 4
Counter has value 5
Counter has value 6
Counter has value 7
Counter has value 8
Counter has value 9
After Loop 10
```

Beware of infinite loops

- All loops must eventually terminate
  - Something inside the loop must eventually make the condition false
- If a loop does not have a way of stopping, it is called an infinite loop
  - An infinite loop continues to repeat until the program is interrupted
- Infinite loops occur when:
  - The programmer forgets to write code inside the loop that makes the condition false.
  - The condition that controls the loop is malformed
- You should avoid writing infinite loops

Examples

```
stars = 1
while  stars <= 10 :
       print ('*', end = ' ')
       stars = stars + 1

#this loop is an infinite loop
stars = 1
while  stars <= 10 :
       print ('*', end = ' ')

#this is an infinite loop too
stars = 1
while  stars >= 1 :
       print ('*', end = ' ')
       stars = stars + 1
```

Note on the while loop

- Note that if the condition is **`False`** the first time you test it, the body of the **`while`** loop may not be executed at all.
- Example:

```
stars = 10
while  stars <= 1 :
    print ('*', end = ' ')
    starts = stars + 1
```

Input Validation

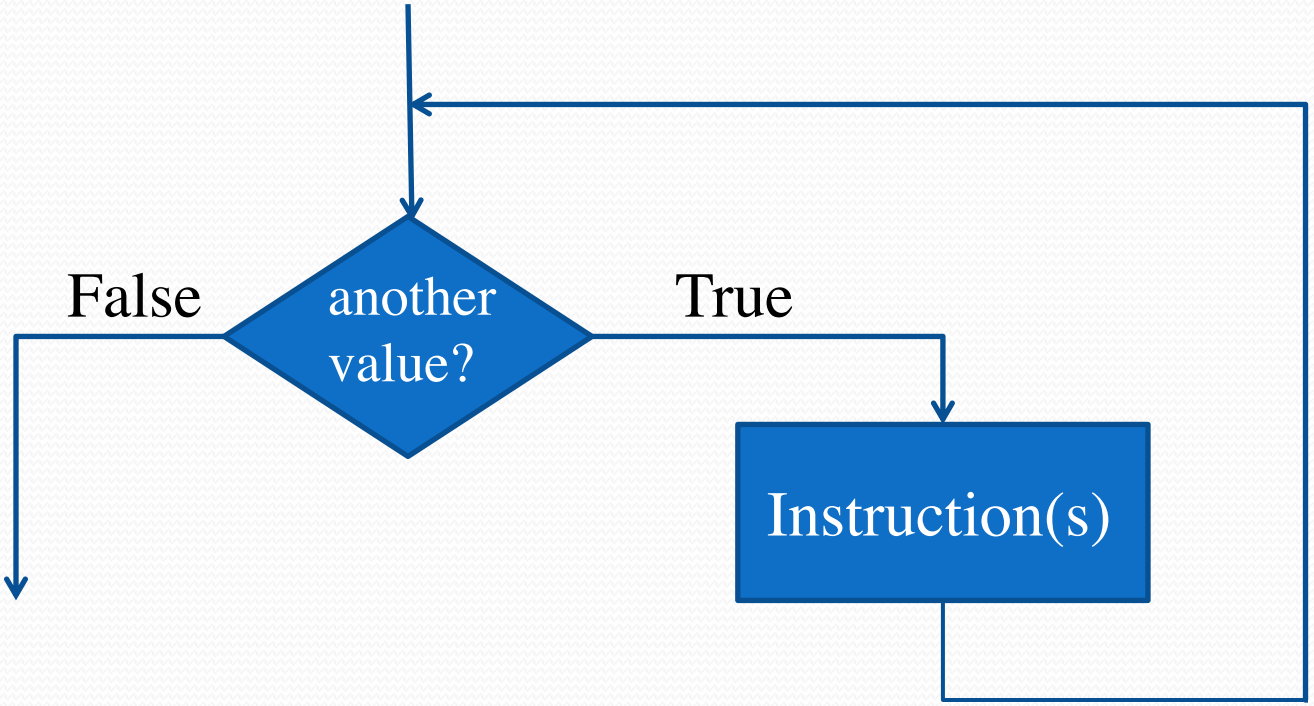
- The **`while`** loop can be used to validate the user's input
- Example:

```
number = int(input('Enter a positive number: '))
while number <= 0 :
    print ('Error: you must enter a value greater than zero. ')
    print ('Try again. ')
    number = int(input('Enter a positive number: '))
```

The for loop

- A count-controlled loop iterates a specific number of times
  - The loop keeps a count of the number of times it iterates
  - When the count reaches a specified amount, the loop stops
- The `for` loop can be used to handle this task
  - We will need to use the `range()` function
  - `range(start, end)` generates a sequence or collection of numbers between `start` and `end`, not including `end`.
  - There is a third optional input: the increment between the elements in the sequence
    - The default increment is 1

The `for` loop



The `for` loop

- Syntax:

```
for value in range(star, end, step) :
    Instruction(s)
```

- Example:

```
for stars in range(0, 10):
    print('*', end = ' ')
```

Output:

```
* * * * * * * * * *
```

for loop and sequences

- We can use **`range()`** to generate the index values for a string, list, or tuple.
- We can combine the **`for`** loop and the index system to access specific parts of the string, list, or tuple.
- We will think of the for loop as: *for each index value in this collection of indices*
- Example

P	y	t	h	o	n
0	1	2	3	4	5

```
string = 'Python'
for index in range(0, len(string)):
    print (string[index], end = ' ')
```

Output:

```
P y t h o n
```

for loop and sequences

Now we can do all sort of things with the characters of our strings, or the elements in our lists, and tuples.

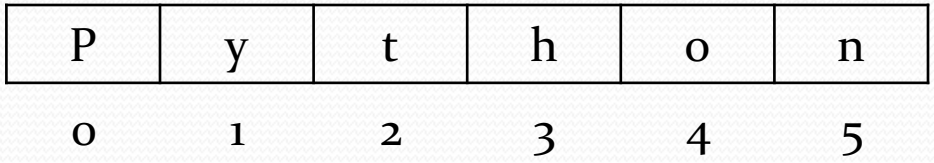
```
for index in range (0 , len(string)//2 ):
    instruction(s)

for index in range (len(string)//2, len(string) ):
    instruction(s)

for index in range (len(string) - 1, -1, -1 ):
    instruction(s)
```

Example

```
#create a mirror of the word Python
string = 'Python'
pile = ''
for index in range(0, len(string) // 2) :
    pile = pile + string [index]
for index in range(len(string) // 2, -1, -1) :
    pile = pile + string [index]
print (pile)
```



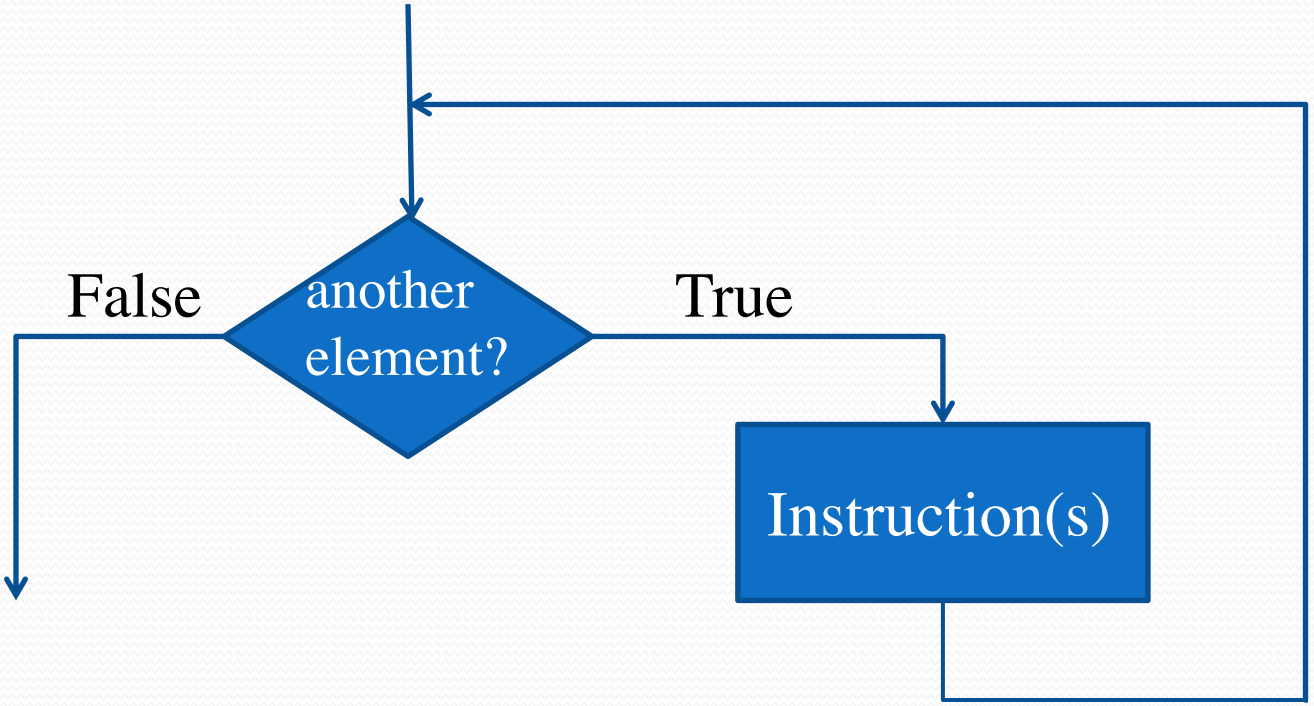
Output:

```
'PythtyP'
```

The `for` loop and collections

- A common programming task is to access all of the elements in a collection/container.
  - Strings, lists, tuples
  - Dictionaries, sets
- A `for` loop can be used to iterate over each element in a container, one at a time
- The container/collection in the `for` loop is typically a list, a tuple, a dictionary, a set, or a string

The `for` loop and collections



The `for` loop and collections

- Syntax:

```
for element in collection :
    Instruction(s)
```

- The next item in the collection is assigned to a variable (`element` in this case) that can be used in the body of the loop
- The `for` loop can be read as "*for each element in this collection*"

Examples

```
string = 'Python'
for char in string:
  print (char)
```

Output:

```
P
y
t
h
o
n
```

```
for char in string :
    print(ord(char), end = ' ‘)
```

Output:

```
80 121 116 104 111 110
```

Examples

```
states = {'New York':'NY', 'New Jersey':'NJ', 'Pennsylvania':'PA'}
for state in states :
    print('The abbreviation for', state, 'is', states[state])
```

Output:

```
The abbreviation for New York is NY
The abbreviation for New Jersey is NJ
The abbreviation for Pennsylvania is PA
```

```
groceryList = ['milk', 'eggs', 'bananas', 'yogurt']
for item in groceryList :
    print('Add', item, 'to the cart')
```

Output:

```
Add milk to the cart
Add eggs to the cart
Add bananas to the cart
Add yogurt to the cart
```

While vs. For -- Guidelines

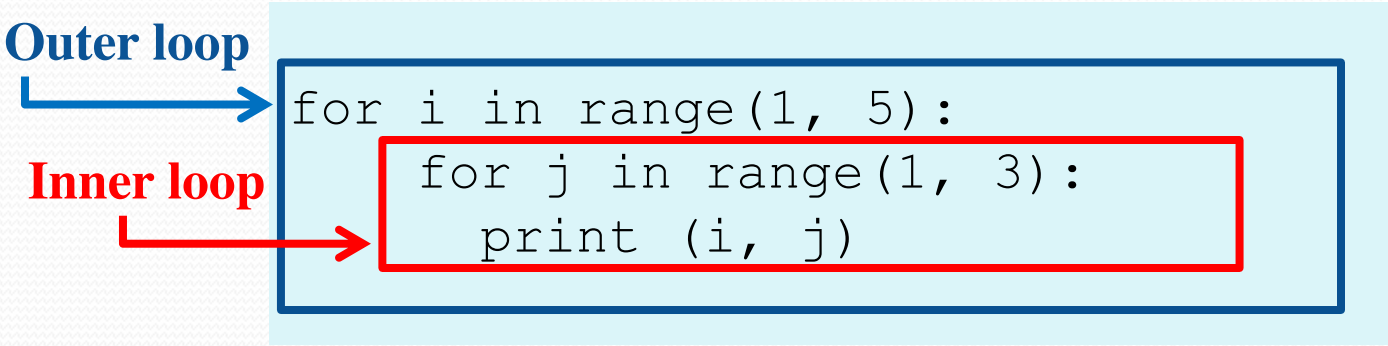
- Use a `for` loop when the number of iterations is computable before entering the loop
  - E.g.: counting down from 10 to 0, printing a string 5 times, etc.
- Use a `for` loop when accessing the elements of a container/sequence
  - Either all of the elements or some of them
- Use a `while` loop when the number of iterations is not computable before entering the loop
  - E.g.: when iterating until a user enters a particular character.

While vs. For -- Guidelines

- The for loop is just a shortcut
- Everything can be done using only while
- It is great to take advantage of built-in shortcuts
- You should practice doing things with a basic while loop
  - Not all Programming Languages have the same features.
  - Everything you might want to do is not built-in.
- You should know how to solve problems with just a while.

Nested Loops

- Nesting is to put one thing inside another
- When we nest loops, we write one loop inside another
- We say then, that the inner loop is nested inside the outer loop



Working with nested loops

- When working with nested loops, we first enter the outer loop for the first iteration
- Once we reach the inner loop, we must work with this loop until we have gone through all its iterations
- Then, once we are completely done with the inner loop, we go back for a second iteration of the outer loop.
- We repeat this process until all the iterations of the outer loop are completed

Understanding nested loops

```
for i in range(1, 5): # [1, 2, 3, 4]
    for j in range(1, 3): # [1, 2]
      print (i, j)
```

- The outer loop will be iterated a total of 4 times
- The inner loop will be iterated 2 times, each time the outer loop is iterated
- Therefore, the inner loop is iterated a total of 8 times

```
Output:     i  j
            1  1
            1  2
            2  1
            2  2
            3  1
            3  2
            4  1
            4  2
```

Example

```
for i in range (1, 6):
    for j in range (1, 5):
        print('*', end = ' ')
    print()
```

Output:

```
* * * *
* * * *
* * * *
* * * *
* * * *
```

Break and Continue

- A `break` statement in a loop causes an immediate exit of the loop.
- A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.
- `break` and `continue` statements can be helpful to avoid excessive indenting/nesting within a loop.
- Caution: because someone reading a program could easily overlook a break or continue statement, such statements should be used only when their use is clear to the reader.

Loop else

- A loop may optionally include an **`else`** clause that executes only if the loop terminates normally, not using a **`break`** statement.
- If the **`else`** statement is used with a **`for`** loop, the **`else`** statement is executed when the loop has exhausted iterating the list.
- If the **`else`** statement is used with a **`while`** loop, the **`else`** statement is executed when the condition becomes false.
- The **`loop else`** construct provides a programmer with the ability to perform some action if the loop completes normally.

Example

```
running = True  #Boolean variable controls the loop
friends = []    #empty list
while running:
    who = input("Who's coming to the party? ")
    if who.lower() == "parents":
        print ('Party plans abandoned')
        break
    if who == "":
        print ('Data entry complete')
        running = False
    else:
        friends.append(who)
else:
    print ('Having a party for', friends)

print('program exiting')
```

Example

```
>>> %Run LoopElse.py
Who's coming to the party? parents
Party plans abandoned
program exiting
>>> %Run LoopElse.py
Who's coming to the party? Ana
Who's coming to the party? Peter
Who's coming to the party? Miguel
Who's coming to the party? Maria
Who's coming to the party?
Data entry complete
Having a party for ['Ana', 'Peter', 'Miguel', 'Maria']
program exiting
```

Computer Programming I

Loop Examples

Asking for input - validation

- A common problem is repeatedly asking for input.
- Loop until the input is valid
  - Data meets a certain criteria
- Use try/except to test data type
  - Data is of the proper data type

Data validation example

```
# Repeatedly ask for input
number = input ('Please enter an integer number: ')
valid = False
while not valid:
    try :
        number = int (number)
        valid = True  # We Succeeded!
    except ValueError as e :
        print("That was not an integer number!")
        number = input ("Please  enter an integer: ")
print('Number doubled is', number * 2)
```

Data validation example

```
Please enter an integer number: one
That was not an integer number!
Please  enter an integer: dos
That was not an integer number!
Please  enter an integer: 3.5
That was not an integer number!
Please  enter an integer: 3
Number doubled is 6
```

Loop until given a certain input

- Example: enter a series of numbers until the user enters *exit*. Compute stats on the values entered and display the results
- Plan:
  - Use *exit* as the sentinel value in the loop
    - Validate input – check that only integers are entered
  - Add given inputs to a list
  - Compute values based on inputs:
    - Min, max, average

Loop until given a certain input

```
print("Enter integer numbers (type 'exit' to stop)")
text = ''
values = []
while text.lower() != 'exit' :
    text = input ("Number: ")
    try :
        number = int(text)
        values.append(number)
    except ValueError :
        print('Not an integer number!')
print('Minimum of Numbers:', min(values))
print('Maximum of Numbers:', max(values))
print('Average of Numbers:', sum(values) / len(values))
```

Loop until given a certain input

```
Enter integer numbers (type 'exit' to stop)
Number: 2
Number: 4
Number: tres
Not an integer number!
Number: 2.3
Not an integer number!
Number: 10
Number: exit
Not an integer number!
Minimum of Numbers: 2
Maximum of Numbers: 10
Average of Numbers: 5.333333333333333
```

Gradebook example

- Problem: simulate a grade-book
  - User should be able to enter as many grades as she/he wishes
  - Grades are integer values between 0 and 100
  - After all grades have been entered, some basic stats need to be calculated: min and max grades, and the average grade
  - We also want to count the number of As, Bs, Cs, etc.
    - A: 90 – 100, B: 80 – 89, C: 70 – 79, D: 60 – 69, F: 0 - 59

Gradebook plan

- For input:
  - Use a sentinel value to stop the look
  - Validate that value entered is an integer between 0 and 100
  - Append each valid value entered to a list
- For stats:
  - We will use a for each style loop to calculate total, min and max.
- For counting letter grades:
  - We will use a dictionary to store letter-grade counts
  - A for loop and conditionals to parse the grade list

- Code posted in Bb Learn