CMPT 120

Topic: Iterative Statements – Part 1

-> for loop
Last Lectures

• Conditional statements
  • if
  • if else
  • Nested if else
  • Chained if (elif)
• Condition: Boolean expression
• Compound conditions
• Short-circuit evaluation of compound conditions
Learning outcomes

At the end of this course, a student is expected to:

• Create (design), analyze, and explain the behaviour of simple algorithms:
  • ...  
  • Design algorithms that avoid the use of break statements to interrupt loops or to avoid the use of multiple return statements to exit a function
• Create (design) small to medium size programs using Python:
  • ...  
  • Refactor repeated statements into for and while loops
Today’s Menu

• Iterative Statements
  • for loop
  • range( ) built-in function
  • in operator
Iterative statements – In real world!

- Imagine that we are having a birthday party
  - 8 guests (including oneself)
- We are serving cake
  - 1 cake -> cut into 8 slices
  - Everybody gets a slice

- While serving a piece of cake to each of our guests, are we repeating some of our actions? Which ones?
What if …

• ... we wanted to do something many times in our program (i.e., execute a set of statements more than once)?
  • How would we go about doing this?

Solution:
1. We could copy and paste a set of statements as many times as required

2. We could make use of yet another Python building block: **iterative statement**
Python statements

- Categories:
  1. Assignment statement
  2. Input statement
     - Conversion function
  3. Output statement
  4. Operational statements
     - Mathematical/arithmetic operators
     - String manipulation operators
  5. Conditional statements
  6. Iterative statements

Some of them are built-in function or method
Iterative statements – In computer world!

6. Iterative Statement
   • **Definition**: Allows us to repeat (iterate) statements in our program

   • **If we know how many times** we want to repeat the statements, then we use the iterative statement called the **for loop**

   • **If we know there is a condition** (or more than 1 condition) that will occur in our program and its occurrence will dictate when we stop repeating a set of statements, then we use the iterative statement called the **while loop**
Back to our Guessing Game

• What if we wanted to allow the user (i.e., player) 3 guesses?
Syntax of a for loop

```python
<statement outside (before) the loop>
for <iterating variable> in <sequence> :
    <first statement to be repeated>
    <second statement to be repeated>
    ...
    <last statement to be repeated>
<statement outside (after) the loop>
```

- Can be a string
- Can be a list
- Can be produced using `range(…)`)
Syntax of a `for` loop

```plaintext
<statement outside (before) the loop>
for <iterating variable> in <sequence> :
    <first statement to be repeated>
    <second statement to be repeated>
    ...
    <last statement to be repeated>
<statement outside (after) the loop>
```

- **Important** – About Indentation
  - *Statements inside the loop* (i.e., statements executed at each iteration of the loop) are the statements indented with respect to the `for` keyword
  - *Statements outside the loop* (before and after the loop) are the statements that are **not** indented with respect to the `for` keyword – these statements are considered to be at the same level of indentation as the `for` loop
Let’s have a look at a for loop

```python
# Magic_for_Loop.py
# <sequence> here is a list

number = 0
aList = [4, 3, 2, 1]
for magic in aList:
    number += magic
print('number = ', number)
```
... and let’s have a look at its **in** operator

```python
aList = [4, 3, 2, 1]
for magic in aList:
    ...
```

Here is how it works:

- For each **element** in the **sequence**
  1. It is assigned to **iterating variable** magic, and
  2. The statement(s) in the body of the **for** loop is/are executed

Here, an element is an integer

Here, the sequence is the list `aList`
Let’s hand trace our example

... so we can figure out how it works

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>number</th>
<th>aList</th>
<th>magic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Another <sequence>: a range

# Range_Example.py
# <sequence> here is a range

define number = 0
stop = 4
for magic in range(stop):
    number += 2
print('number = ', number)
Built-in function `range()`

- Very useful in `for` loop
- **Syntax:** `range([start,] stop [,step])`
- Produces a list of integers
- **How does it work?**
  - If `theLength = 5`
  - Then `range(theLength)` produces `[0, 1, 2, 3, 4]`
Hand tracing our example

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>number</th>
<th>range(stop)</th>
<th>magic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let’s play around with `range()`

- What will we see printed on the Python Interpreter shell window if:
  - `range(10)`
  - `range(1, 11)`
  - `range(0, 30, 5)`
  - `range(0, 10, -3)`
  - `range(0, -10, -1)`
  - `range(0)`

- To see it on the Python Interpreter shell window:
  - `list(range(...))`
Yet another `<sequence>`: a string

```python
# Count_Letter_2.py
# `<sequence>` here is a string

# Set variables
count = 0
fruit = "banana"

# Loop - Spell a word and count the number of letter in the word
# Running count
for letter in fruit:
    count += 1
    print("Letter %d is '%s'")%(count, letter)

# Output result
print("The string '%s' has %d letters.")%(fruit, count)
```

Hand tracing our example

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>count</th>
<th>fruit</th>
<th>letter</th>
</tr>
</thead>
</table>

[20]
Let’s Practise

• **Problem Statement:**
  • Create a Python program that spells the word the user enters and prints its length

• **Sample Run:**

  Please, enter a word: apple
  The word you entered is apple
  letter 1 is: a
  letter 2 is: p
  letter 3 is: p
  letter 4 is: l
  letter 5 is: e
  This word has 5 letters.
Solution – Word_1.py

aWord = input("Please, enter a word: ")
print("The word you entered is '%%s'" %aWord)
theLength = len(aWord)

# Way 1:
for index in range(theLength):
    print("Letter %i is: '%s'
        %(index+1,aWord[index])")

print("This word '%s' has %i letters."
    %(aWord, theLength))
Let’s hand trace Word_1.py

| Iteration # | aWord | range(theLength) | index |
Solution – Word_2.py

aWord = input("Please, enter a word: ")
print("The word you entered is '%s'" %aWord)
theLength = len(aWord)

# Way 2:
index = 1
for letter in aWord:
    print("Letter %i is: '%s'" % (index, letter))
    index = index + 1

print("This word '%s' has %i letters." % (aWord, theLength))
Let’s hand trace `Word_2.py`

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>aWord</th>
<th>theLength</th>
<th>index</th>
<th>letter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Back to our Guessing Game

• What if we wanted to allow the user (i.e., player) 3 guesses?

• Solution: GuessingGame_5.py

• Step 5 – Testing
  • How would we test this program?
Summary

• Iterative Statements – Part 1
  • for loop
  • range() built-in function
  • in operator
Next Lecture

• Iterative Statements – Part 2
  • while loop