CMPT 120
Intro to CS & Programming I

WEEK 5 (Feb. 3-7)

— Jérémie O. Lumbroso —

Lecture 12:
More about Variables in Python, and Some Code Templates

http://www.sfu.ca/~jlumbros/Courses/CMPT120/

Let's see once more how Python variables work

PYTHON VARIABLES

Variables

- Variables are used to
  - store intermediate results
  - transmit results (for instance in functions)

```
result = result + x
```

```
def someFunction(varA):
  ...
```

Syntax

(You already know this, but to refresh your memory!)

- Assign: `myvar = <some expression>`
  - the expression is first computed
  - if there is no error, the result of the computation is placed inside the box `myvar`
- Reference: `myvar`
  - can be used in an expression, can be used anywhere a literal value would be used
  - if a variable is referenced before it is defined for the first time, there is a Python error, `NameError: name 'myvar' is not defined`

Order of Evaluation

- The statement `x = f(y+z) + g(z+w, z+y)` is computed as follows
  - first `f` is looked up (to see if the function is defined)
  - then `y` is looked up (to see if the variable is defined)
  - then `z` is looked up
  - then `y+z` is evaluated, if they both exist
  - then `f` is called with the argument equal to `y+z`
  - then all this is done with the call to `g`
  - then once `f(y+z)` and `g(z+w, z+y)` are computed, the expression `f(y+z) + g(z+w, z+y)` is computed
  - if there is no error, then the result of this expression is placed into the box called `x`

Some typical usages of variables

SOME TEMPLATES

Several “Routine” Uses

- In the following slides, we will see how a variable is used
  1. to accumulate a result (sum of range)
  2. to save the last result found (last occurrence)
  3. to keep track of a flag (contiguous spaces)

Accumulating Variable

- A variable can be used to "accumulate" a result, for instance
  - when you have a sum of many numbers, you add them each one by one to the variable
  - when you have a product of many numbers, you multiply the variable by each of them one by one
  - when you have a maximum of many numbers, etc.
  - when you have a minimum of many numbers, etc.

Accumulating Variable Scheme

- There are two components to this type of scheme
  - what the initialization value of the accumulation variable is?
  - how to “incorporate” a new element into the variable

Sum of Range (Yet Again!!)

- Initialization value: `0`
- Combination is: `sumvar = sumvar + i`

```
# sumRange returns the sum of first+(first+1)+...+(last-1)+last.
# Hypotheses: first and last are integers, first <= last.

def sumRange(first, last):
  sumvar = 0
  for i in range(first, last+1):
    sumvar = sumvar + i
  return sumvar
```

Another Example

- Would this work for a minimum (by switching the sign from < to > in the if statement)?
- What would you have to pick as an initialization value?
- Choice of initialization value is **important**

```python
import random

# This function returns the maximum of num_of_ints
# random integers drawn between 0 and 100.

def maxOfRandomInts(num_of_ints):
  maxvar = 0

  for i in range(num_of_ints):
    # We draw a random number between 0 and 100
    mynum = random.randint(0,100)

    # Accumulate
    if maxvar < mynum:
      maxvar = mynum

  return maxvar
```

Finding **First** Occurrence in String

- Finding the first occurrence of a character in a string
- Does not require an auxiliary variable

```
# Finds the position of the *first* occurrence
# of letter in phrase.

def findFirstOccurrence(phrase, letter):
  for i in range(len(phrase)):
    if phrase[i] == letter:
      return i
  return -1
```

Finding **Last** Occurrence in String

- Using return the first time a character is found would not work here
- We need to keep track of the position of last found character, and only at the end, return the last one found

```
# Finds the position of the *last* occurrence
# of letter in phrase.

def findLastOccurrence(phrase, letter):
  most_recent_position = -1
  for i in range(len(phrase)):
    if phrase[i] == letter:
      most_recent_position = i
  return most_recent_position
```

Comparison

First occurrence

- No auxiliary variable
- As soon as occurrence found, the return statement exits the function: thus no other occurrences are even examined (if they exist)
- After the loop, if the function is still being exited then no occurrence was found: so return statement with default value

```
def findFirstOccurrence(phrase, letter):
  for i in range(len(phrase)):
    if phrase[i] == letter:
      return i
  return -1
```

Last occurrence

- Auxiliary variable stores the most recently found position
- At end of loop, the variable contains the last position
- Auxiliary variable initialized with default value
- If variable never updated (no occurrence found), it will still contain this default value at the end of the loop

```
def findLastOccurrence(phrase, letter):
  most_recent_position = -1
  for i in range(len(phrase)):
    if phrase[i] == letter:
      most_recent_position = i
  return most_recent_position
```

Flag Variable

- A flag variable is usually set to a state to **flag** what is happening, for instance

```
def removeContiguousSpaces(phrase):
  result = ""
  flag = 0

  for ch in phrase:
    if ch == " ":
      if flag == 0:
        # We see the first space in a possible set.
        result = result + ch
        flag = 1
      # If not, then the flag is set to 1, so we do not
      # need to duplicate a space
    else:
      # The ch character is not a space, so we reset
      # the flag (since the flag marks when we have just
      # seen a space).
      flag = 0
      result = result + ch

  return result
```