CMPT 120
Intro to CS & Programming I

WEEK 6 (Feb. 17-21)

— *Jérémie O. Lumbroso* —

Lecture 14:
More examples using Flags and Strings + Nested For Loops

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

A few useful input functions

INPUT FUNCTIONS

Ask user for specific input

- Follows pattern of Ex. 2 Q3, `ask_number()`
- Four steps
  1. Ask user for input following a certain format (integer, "yes" or "no", a selection from a list)
  2. Check that the input is correct
  3. If it is, then return it as a result of your function
  4. If not prompt the user to try again (or inform him a *default choice* has been made for him)

How to "try again"/restart everything?

- When user gives wrong input, how do you have him try again?
- The function is defined: you just need to have it call itself
- … and **make sure to return the result**

```
def ask_number():
  # All the code that does everything
  # ...
  # ...

  if inputCorrect:
    # Do what you normally would do
    # ..

    return something

  else:
    # If the input is incorrect,
    # restart function, and return what
    # the new call to the function
    # will return

    return ask_number()
```

`ask_bool()`

- Function to ask "yes" or "no" from user
- Accept different cases ("yes","YES","YeS", …)
- Accept shorthands ("Y","y","N","n")
- Return a boolean: `True` or `False`

Optional

- Variations? ("never" = "no","always" = "yes")
- Jokes? ("maybe")

ask_bool()

```
def ask_bool():
  input = raw_input("[Y]es or [n]o? ")

  # Lower the case to have less to check
  input = input.lower()

  if input == "yes" or input == "y":
    return True

  # IMPORTANT: note "elif" -> "else if"
  elif input == "no" or input == "n":
    return False

  else:
    if input == "maybe":
      print "What are you, an eight ball?"

    print "Input incorrect, try again."

    # Call the function itself and return
    # that result.
    return ask_bool()
```

http://goo.gl/r0jHa1

Where can we go wrong?

- When you call the function itself to try again, you must `return` the result, or else the result will be lost

```
def ask_bool():
  input = raw_input("[Y]es or [n]o? ")

  # Lower the case to have less to check
  input = input.lower()

  if input == "yes" or input == "y":
    return True

  # IMPORTANT: note "elif" -> "else if"
  elif input == "no" or input == "n":
    return False

  else:
    if input == "maybe":
      print "What are you, an eight ball?"

    print "Input incorrect, try again."

    # Call the function itself and return
    # that result.
    ask_bool()
```

Difference between `elif` and `else`-newline-indent-if

```
def ask_bool():
  input = raw_input("[Y]es or [n]o? ")

  # Lower the case to have less to check
  input = input.lower()

  if input == "yes" or input == "y":
    return True

  # IMPORTANT: note "elif" -> "else if"
  elif input == "no" or input == "n":
    return False

  else:
    if input == "maybe":
      print "What are you, an eight ball?"

    print "Input incorrect, try again."

    # Call the function itself and return
    # that result.
    return ask_bool()
```

When input is incorrect, we restart, but we also do something special when "maybe" is the incorrect input.

```
def ask_bool():
  input = raw_input("[Y]es or [n]o? ")

  # Lower the case to have less to check
  input = input.lower()

  if input == "yes" or input == "y":
    return True

  # IMPORTANT: note "elif" -> "else if"
  elif input == "no" or input == "n":
    return False

  elif input == "maybe":
    print "What are you, an eight ball?"

    print "Input incorrect, try again."

    # Call the function itself and return
    # that result.
    return ask_bool()
```

There are three cases: "yes"/"y", "no"/"n", and "maybe". The last `elif` treats the "maybe" case, but all other incorrect input is not covered.

Course Exercise 4 [*Practice*]

- The user is going to think of a number between 1 and 100
- Write program that is going to guess it
- Stupid program:

```
def stupid_number_guess(smallest, largest):
  print "Think of a number between", smallest, "and", largest

  # Just go through every number one by one
  for num in range(smallest, largest+1):
    print "Is your number", num,
    if ask_bool():
      # If the user answers "Yes" (and ask_bool() returns True)
      # then we print a message and return to exit the function
      print "I have guessed your number! It is", num
      return

  # If we have gone through all numbers between smallest
  # and largest, then the user must have not correctly
  # answered a question, or his/her number is out of range
  print "You either made a mistake or your number is out of range."
```

http://goo.gl/9kVklb

An advanced example using two flags

ADVANCED FLAGS

Flag Variables

- Inside a function, or loop, we might have to detect the state of a condition
- "Flag" variable: detect when a condition changes and we need to **remember**
- For example: finding if someone on the first row is wearing glasses

Very Advanced Example 1

- Print all words in a sentence that start with an uppercase letter
- Words: continuous strings of alphabetical characters, use `isalpha()`
- Upper: THIS IS UPPERCASE, use `isupper()`
- Two flags:
  - One flag to **remember** if we are building a word
  - One flag to **remember** if first letter of a word is uppercase

Uppercase Words

- Two flags
  - `encountered_new_word` determines whether we are in the middle of a block of alphabetical characters
  - `isfirstupper` determines if first character is uppercase
- Print a word only when we encounter a non-alphabetical character when the flag `isword` is set to `True` (and only if isfirstupper is **also** `True`)
- *Examine this example closely*

```
def uppercase_words(mystring):
  encountered_new_word = False  # Flag when building a word
  isfirstupper = False  # Flag to see if first letter is uppercase
  myword = ""           # Buffer containing the word

  # Little trick: we add a non-alphabetical character at the end
  # of the string (QUESTION: why?)
  mystring = mystring + "@"

  for ch in mystring:
    if ch.isalpha():
      # The character is alphabetical

      if not encountered_new_word:
        # The previous character was not alphabetical
        # so this is the first letter of the word
        encountered_new_word = True
        isfirstupper = ch.isupper()

        # We also reset the word's buffer
        myword = ""

      # In all cases (whether the character is the first of
      # the word or not), we add the character to the buffer
      # string
      myword = myword + ch
    else:
      # The character is non-alphabetical
      if encountered_new_word:
        # If we were building a word, now we are not any longer
        encountered_new_word = False
        if isfirstupper:
          print "Capitalized word:", myword
```

http://goo.gl/03D9aU

A loop inside a loop

NESTED FORS

Nested For Loops

- Several for loops can be nested inside each other
- **Be careful to use different variables** for the index of each loop
- Print all pairs (1-10), (1-11), … (2-10), … (3-10), (4-11), …, (4-13)

```
for i in range(1,4):
  for j in range(10,14):
    print "(", i, "-", j, ")  "
```