# A201/A597 Introduction to Programming I

## First Summer 2007

Lecture Six: Tuesday May 15, 2007 (ED1204)

Announcement: first week's assignment (comprising labs 1, 2, 3 and homework 1, 2) is due today in OnCourse. Today in lab we will start by making sure we're all clear on that.

Yesterday we discussed only one problem:

1. Write a program that accepts any number of lines from the user, one at a time, and prints them back. When the user enters the string `bye` the program ends and prints the number of lines the user has entered up to that point (with the exception of `bye`).

Here's how our discussion went. We started from this simple code snippet:

```
i = 4

if i < 10:
  i = i + 1
```

It was very easy to describe the behaviour of this program: i starts by being 4 and because that is less than 10 i becomes 5. Kind of quiet, but that's what the program does. Can we make this program be a little more outspoken and produce some output? Here's a change:

```
i = 4

if i < 10:
  i = i + 1
  print i
```

Now the program also prints the value.

What if we make the following change:

```
i = 4

while i < 10:
  i = i + 1
  print i
```

Now the variable will be incremented and printed 6 times.

That was our introduction to while-loops.

We also noted that the code below prints only the last value, 10, for which the loop ends.

```
i = 4
while i < 10:
  i = i + 1
print i
```

Similarly, it was easy to determine the effect of these two modified snippets.

```
i = 14

while i < 10:
  i = i + 1
  print i
```

The code above skips the loop altogether, while the code below is an infinite loop.

```
i = 4

while i < 10:
  i = i - 1
  print i
```

With this we started to discuss the solution to the first problem.

```
count = 0
line = raw_input("Enter: ")
while line != "bye":
  print line
  count = count + 1
print "Thank you for using this program.
print "You have typed", count, "lines. "
```

This looked almost perfect but had a flaw.

Do you remember what it was (or determine what it is)?

The flaw was easy to fix:

```
count = 0
line = raw_input("Enter: ")
while line != "bye":
  print line
  count = count + 1
  line = raw_input(“Enter: “)
print "Thank you for using this program.
print "You have typed", count, "lines. "
```

With this behind us the next four programs are your lab assignment for today.

Lab Assignment Four (due with all of this week’s work on Tue May 22,in OnCourse).

2. Write a program that creates random addition questions (by generating random integer operands between -50 and 50) and asks the user to solve them. Your program should keep track of the number of good answers as well as of total questions asked. Write two such programs: one that asks ten questions then ends, and another one that ends only when the user types `bye`. Either way the program should report a score at the end.

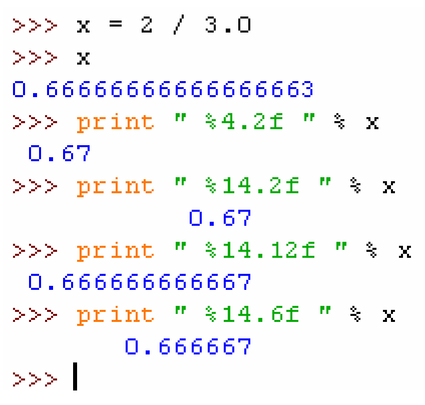
3. Write a program that randomly picks an integer between 0 and 100 and then asks the user to guess it. The game ends when the user guesses the number. Each time the user enters a guess value feedback is provided (try higher, try lower) and a count is kept that records the number of incorrect guesses made up to that point. Report this number when the game ends.

4. Modify the program above so that if the user makes 6 wrong guesses the game is lost.

5. Write a program that expects the user to enter a number per line and prints back (after every line) the current average of all the numbers entered up to that point. Program stops when the user enters the string `bye`.

Today we will also discuss number formatting by looking at

```
http://docs.python.org/lib/typesseq-strings.html
```



```
>>> x = 2 / 3.0
>>> x
0.66666666666666663
>>> print " %4.2f " % x
 0.67
>>> print " %14.2f " % x
           0.67
>>> print " %14.12f " % x
 0.666666666667
>>> print " %14.6f " % x
       0.666667
>>> 
```

Lab Four: Tuesday May 15, 2007 (ED2025)

The lab assignment being stated above, the job of the notes below is to help you organize yourself when reading and experimenting with the material in chapter 3. Thus, it follows the text closely but gives an example of taking notes by creative experimentation.

In this chapter you will learn how to selectively execute certain portions of your code and repeat parts of your program. The chapter starts by describing the “guess my number” game which is also our problem no. 3 above (perhaps with the extra feature from no. 4).

Generating random numbers is explained on pp. 52-54. There are two steps to it:

a) first import the random module
b) when a random number is needed call random.randrange(…)

Important note: number generated this way are integers, between 0 and arg-1 where arg is the argument value passed to randrange.

Branching is a fundamental part of computer programming. It basically means making a decision to take one path or another. Recall our minute paper of yesterday whose purpose was to read a number from the user and report whether it was odd or even:

```
num = int(raw_input("Enter: "))

if num % 2 == 0:
  print “The number”, num, “is even.”
else:
  print “The number”, num, “is odd.”
```

Note that the code for the two branches is indented. The end of indentation represents the end of the code for that branch. (Our branches are very short, just one line each).

All if structures relies on conditions. Conditions evaluate to truth values: their type is boolean. Comparison operators can be used with numbers to create conditions: <, <=, >, >=, ==, `!=`. They are also called relational operators (table on page 58, at the top).

The code for one branch is called a block. Using indentation to create blocks is the topic of page 58. The most common type of conditional structure is the if-else structure (the one we used in the minute paper program of yesterday, see code above). Other types include the if without else and the if-elif-else structure. This last structure was used in the letter grade program for lab assignment three. Let’s work out another example, similar to the Mood Computer Program of pp. 61-63.

Write a program that can print one fragment (lyrics) from any of four songs that you enjoy. The choice is the user's who can pick one of the four songs and have the chosen lyrics printed. The example below assumes you are an R.E.M. fan:

```
>>>Please choose from the menu below:
  [1] The Great Beyond
  [2] Fretless
  [3] Losing My Religion
  [4] It's a Free World, Baby
What would you like to listen to?
Please enter a number: 1

I've watched the stars fall silent from your eyes
All the sights that I have seen
I can't believe that I believed I wished
That you could see
There's a new planet in the solar system
There is nothing up my sleeve

>>> Please choose from the menu below:
  [1] The Great Beyond
  [2] Fretless
  [3] Losing My Religion
  [4] It's a Free World, Baby
What would you like to listen to?
Please enter a number: 3

Life is bigger
It's bigger than you
And you are not me
The lengths that I will go to
The distance in your eyes
Oh no I've said too much
I set it up

>>> Please choose from the menu below:
  [1] The Great Beyond
  [2] Fretless
  [3] Losing My Religion
  [4] It's a Free World, Baby
What would you like to listen to?
Please enter a number: -1
That's not a valid choice.

>>>
```

The code for this program is very straightforward:

```
choice = int(raw_input("Please choose from the menu: ... "))
if choice == 1:
  print "The Great Beyond"
elif choice == 2:
  print "Fretless"
elif choice == 3:
  print "Losing My Religion"
elif choice == 4:
  print "It's a Free World, Baby"
else:
  print "I am afraid that is not a valid option... "
```

The branching structures discussed are summarized in a table on page 65.

Minute paper for today is listed here: consider the two code fragments below.

```
if x == 5:
  x = x + 1
else:
  x = 8
```

```
if x == 5:
  x = x + 1
if x != 5:
  x = 8
```

The question is: are the two fragments equivalent?

Next we start discussing while loops.

The three year old simulator program on page 66 illustrates the basic dialog loop with a user: while the line read from the user does not match a key word the program keeps asking for a new line. The prompt of the program is the string "Why?" which is why the program is said to emulate a three year old. The keyword to end the conversation with the program is "Because." Note that it has to be typed with this exact capitalization, including the period at the end, otherwise the program doesn't recognize it.

The program we developed yesterday in class is another example of this type of program.

The discussion in the book (pp. 66-68) examines the while structure and defines the concept of a sentry variable. A sentry variable is a variable used in a condition of a while loop and compared to some other value or values. Like a human sentry, you can think of your sentry variable as a guard, helping form a barrier around the while loop's block. It is important to initialize your sentry variable (p. 67).

It is also important to update your sentry variable if your program relies on that. This is precisely what we forgot in the first version of the program we wrote yesterday. If your sentry variable should be updated but isn't (by mistake) you have an inifinite loop. There's no other way to protect against unwanted infinite loops except for clear thinking.

The program on page 69 (the Losing Battle program) resembles this one below:

```
x = 7
while x > 1:
    if x % 2 == 0:
        x = x / 2
    else:
        x = 3 * x + 1
    print x
print x
```

Trace the program for other initial values of x, such as: 16, 10, 27, 3, etc.

Like Losing Battle the code above relies on a condition that can become False.

Next, on pp. 72-74 we discuss the special nature of boolean values in Python. The entire discussion is summarized on page 74 at the top: the rules for what makes a value True or False are simple. The basic principle is this: any empty or zero value is False, everything else is True. The empty string “” is False, while any other string is True (even a string made only of spaces). This simplifies conditions most often than not.

The loop features break and continue are next, in the context of infinite loops.

One can use break to terminate an otherwise infinite loop.

Thus the program we wrote yesterday could be:

```
count = 0
while True:
  line = raw_input(“Enter: “)
  if line != "bye":
    break
  else:
    print line
    count = count + 1
print "Thank you for using this program.
print "You have typed", count, "lines. "
```

The other statement, continue, has a different role. It resumes the loop. Thus, we could change the program one more time so lines that read “silent” are not counted. Here’s how the program becomes:

```
count = 0
while True:
  line = raw_input(“Enter: “)
  if line != "bye":
    break
  elif line == “silent”:
    continue
  else:
    print line
    count = count + 1
print "Thank you for using this program.
print "You have typed", count, "lines. "
```

So if we don’t stop and we don’t skip we print and count the line.

Exercise: write a program that accepts any number of integers from the user, one per line, terminated with “quit” and reports their average. Numbers outside the range [0, 100] are not to be included in the average calculation.

Sooner or later we need to discuss compound conditions. They rely on boolean operators: not, and, or. We discussed them in class. Their truth tables are listed on pp. 78-81. Here are a few more exercises like the ones discussed in class. Given:

| `p` | `q` | `p` **`and`** `q` | `p` **`or`** `q` | **`not`** `q` |
|---|---|---|---|---|
| `true` | `true` | `true` | `true` | `false` |
| `true` | `false` | `false` | `true` | `true` |
| `false` | `true` | `false` | `true` | |
| `false` | `false` | `false` | `false` | |

- can you simplify `p == True`?
- how do you simplify `p == False`?
- is is true that `not (p and q) == not p or not q`?
- same question for `not (p or q) == not p and not q`.

Note that not is like a unary minus, while and is like a multiplication and or is like a plus.

This has serious implications for the order of evaluation.

Thus, a or b and not c is evaluated as follows: a or (b and (not c)).

Some more questions: what does the following print?

```
if False and False or True:
    print "False"
else:
    print "True"
```

What if we add parens like so:

```
if False and (False or True):
    print "False"
else:
    print "True"
```

Can you calculate the value of `a or not a` if a is a boolean variable?

Here's how we do it:

| `a` | `not a` | `a or (not a)` |
|---|---|---|
| `true` | `false` | `true` |
| `false` | `true` | `true` |

So we see the value does not depend on a and truth tables are the way to go.

What is the truth table for `not a or not b` ?

Let's build it at the same time with the one for `not(a and b)`

| a | b | a and b | not (a and b) | not a | not b | not a or not b |
|---|---|---|---|---|---|---|
| true | true | true | false | false | false | false |
| true | false | false | true | false | true | true |
| false | true | false | true | true | false | true |
| false | false | false | true | true | true | true |

We have just proved one of DeMorgan's law.

What is the other one?

It's the dual of this: `not(a or b)` is the same as `not a and not b`

There are many other identities that one can prove.

Perhaps we can do that later, as needed.

Yes, but let me give some examples, in case you get bored and want to practice.

Sure.

| This | ... is the same as this |
|---|---|
| `a and (b or c)` | `a and b or a and c` |
| `a or true` | `true` |
| `a and true` | `a` |
| `a or false` | `a` |
| `a and false` | `false` |
| `a == true` | `a` |
| `a == false` | `not a` |

Let's face it: `boolean`s can make you dizzy.

Yes, but they are clearly necessary.

The last thing in this chapter refers to design.

We will address that issue when we develop the programs in our lab assignment.

Homework Three

A year with 366 days is called a leap year.

A year is a leap year if

- it is divisible by 4 (for example, the year 1980),
- except it is not a leap year if it is divisible by 100 (for example, the year 1900);
- however, it is a leap year if it is divisible by 400 (for example, the year 2000).

There were no exceptions before the introduction of the Gregorian calendar on October 15, 1582 (for example, the year 1500 was a leap year).

Write a program (called `leap.py`) that asks the user for a year and computes whether that year is a leap year or not.

Here's a succesion of sample runs for such a program:

```
Please enter the year then press Enter : 1500
Leap year: 1500

Please enter the year then press Enter : 1900
1900 not a leap year!

Please enter the year then press Enter : 1996
Leap year: 1996

Please enter the year then press Enter : 1997
1997 not a leap year!

Please enter the year then press Enter : 2000
Leap year: 2000
```