CSC108: Introduction to Computer Programming

Lecture 1

Wael Aboulsaadat

Acknowledgment: these slides are based on material by: Velian Pandeliev, Diane Horton, Michael Samozi, Jennifer Campbell, and Paul Gries from CS UoT

Welcome!

- No prerequisites, no previous experience required

- Course objectives: Understanding fundamental programming principles, combining them to generate solutions to interesting problems and writing programs in Python

- Topics: variables, functions, conditionals, loops, debugging, testing, text file processing, dictionaries, sorting, algorithm design, image manipulation

Recommended Text Book

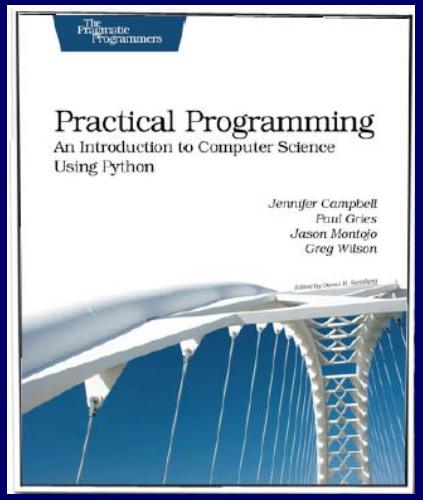
Practical Programming

An Introduction to Computer Science

Using Python

J. Campbell, P. Gries, J. Montojo, G. Wilson

Publisher: Pragmatic Programmers 2009



Website

- CSC108 is hosted on Blackboard (www.portal.utoronto.ca)
- You can login with your UTORid and password
- There you will find:
  - Lecture notes
  - Labs and Assignments
  - Discussion Boards

Evaluation

- 3 Assignments → 30%
- 1 Midterm → 10%
- 4 Quizzes → 10%
- 10 Labs → 5%
- 11 CodeLabs → 5%
- Final Exam → 40%

100% !

Labs (5%) and CodeLabs (5%)

- Labs are weekly two-hour practice sessions in groups of 20-25. Marks are earned through attendance and effort. They start next week: check website for room assignment.

- CodeLabs are small online exercises due every Monday morning by the beginning of class. They are found at www.turingscraft.com

- Register for CodeLabs online with your **real name** and U of T e-mail address. Cost: $25 US

Assignments (30%)

- All involve writing Python programs
- Submitted electronically by 10 pm on the due date. No late assignments will be accepted.
- You can work in pairs on Assignments 1 and 3
- Assignment 2 must be done individually
- You must work with different partners on Assignment 1 and Assignment 3

Quizzes (10%), Midterm (10%), Exam (40%)

- Four 15-minute single-question quizzes will be spread out over the term
- There's a 50-minute in-class midterm
- The 3-hour final exam will be written in the April exam period.
- You need 40% on the exam to pass the course.

Administrative Notes

- There is no make-up date for the midterm. If you're sick or away for a legitimate reason, other course components will be applied to your midterm mark.

- If you are sick, you need a doctor's note from the University clinic for any accommodations to be made.

- Re-marking: if you believe there's a reason why your work should be looked at again, you have 7 days from the time your work is returned to you to file a request (form is on the site).

Computing on Campus

- Computing Discipline Facility (CDF)
- Bahen Centre (BA3200, BA2200, BA2210, BA2220, BA2240, BA2270), Gerstein Centre (2nd floor)
- BA3175, BA3185, BA3195 are also available, but they're often reserved for labs
- To login to CDF, find your username at http://www.cdf.toronto.edu/cgi-bin/webfinger
- Your password is initially your student number

Getting Help

- Labs (practice)
- Office hours (Wed 3–5 in Bahen 4261)
- DCS Help Centre (BA2200, 4-6 pm Mon-Thurs)
- Online discussion boards (for general questions)
- E-mail: wael@cs.toronto.edu (for personal matters)

Academic Offences

- Never share solutions with another person or group
- Discuss assignment solutions only with course TAs and instructors
- Keep your eyes on your own test
- Do not use code you didn't write

In Class

- I will teach and pause for questions
- Let me know if I'm going too quickly or too slowly
- Be respectful and considerate of others

Computers and Programming

Programs

- A program is a set of instructions that a computer understands and executes.

- We write programs to accomplish tasks or solve problems.

- There are two steps to writing a program:
  1) Devise an **algorithm** **that solves the problem** (language-independent)
  2) Come up with the appropriate sequence of instructions to execute (language-specific)

E.g. The average of 5 numbers

- **Problem:**

  Given five numbers, find their average.
  - What's the algorithm for solving this problem?
  - And the program?

Programming Languages

- Programming languages are artificial languages that enable humans to convey sequences of instructions to a computer.

- Like natural languages, they have rules, grammar and syntax.

Computers and Computation

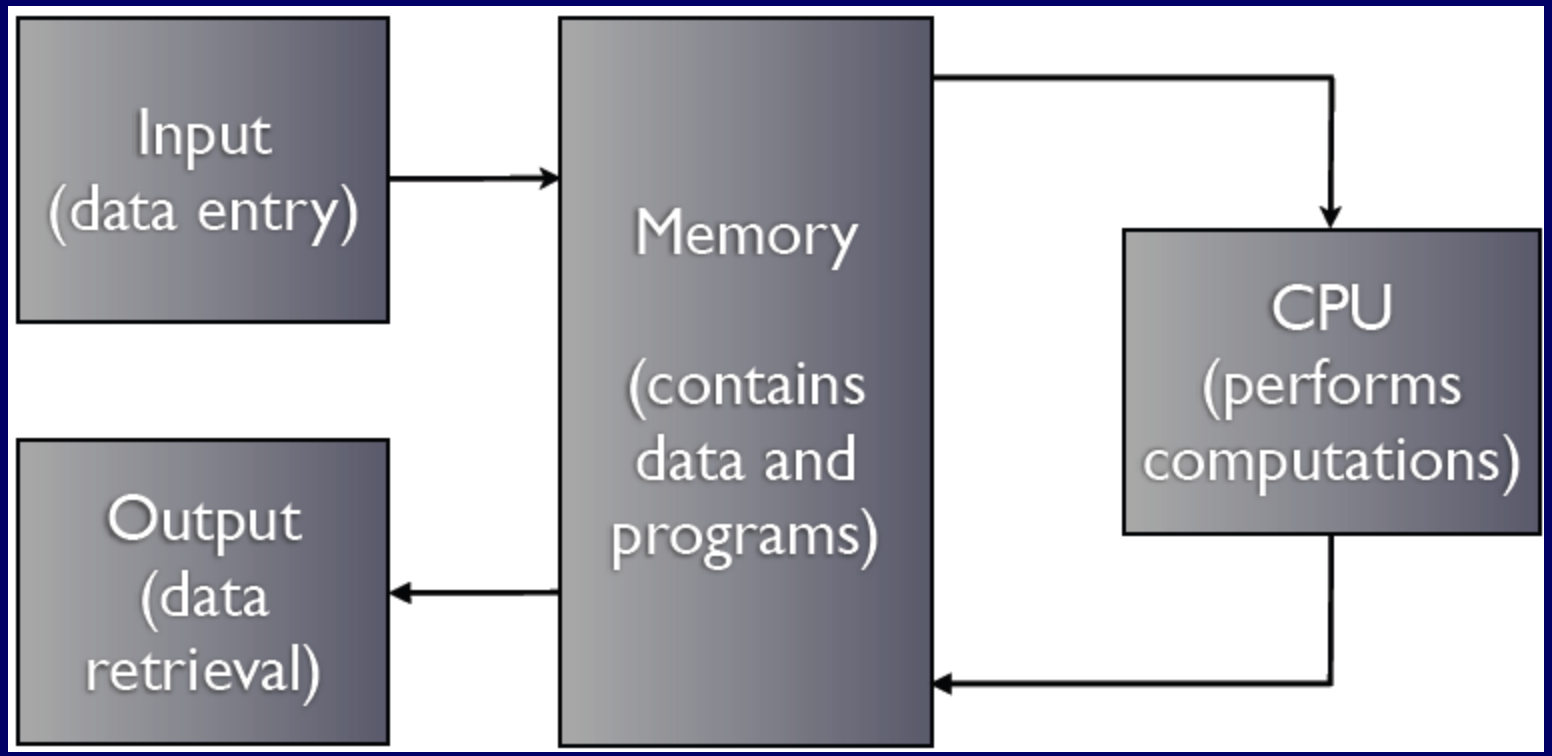
- We know two discordant things about computers:

1) They are incredibly powerful and have applications in every avenue of human activity

2) They are fast glorified calculators that work with strings of 1's and 0's

1100010101010101001010100101010010010100
0101010100111011010101010….

Computer Architecture



Memory

- Computer memory is binary (uses 1's and 0's to encode information)
- A single binary digit is called a **bit**
- Every piece of data and every instruction in memory ends up as a binary string

CPU

- Every CPU has a rigid set of (about 50) operations it can perform, including arithmetic operations and memory manipulation

- Each of these operations has a unique binary code (known as an **opcode) that the CPU recognizes**
  - E.g. The bit string 00000011 signals an addition

- The CPU reads programs from memory and performs computations according to the sequence of opcodes it encounters

Language Abstraction

- To be executed, programs have to end up as binary strings that the CPU can read
- In theory, one could write any program just by manipulating the 1's and 0's stored on a computer's hard drive
- However, programmers would have to remember a lot of binary opcodes and mistakes would be very hard to catch

Low-Level Languages

- In low-level languages (also called assembly or machine languages), every operation the CPU can perform has a mnemonic code, e.g. ADD for addition, MOV for moving values
- Low-level code is automatically converted to binary before being fed to the CPU
- Low-level languages are **hardware-specific -** every CPU architecture uses a different one

Low-Level Languages

- Low-level code is more readable than binary machine code, but not by much
- Every low-level instruction corresponds to a single CPU operation
- More complex tasks require sequences of low-level instructions which often appear together, but have to be typed out individually
- With low-level code, the programmer has to manage
  - memory manually, keeping track of memory, addresses etc..

High-Level Languages

- High-level languages were created to make programming a simpler task for the end user
- Groups of low-level operations can be expressed with a single high-level command
- Many high-level languages automate memory management
- High-level languages are **hardware independent**

Example: A single instruction

- In binary:
  - 11000111000001100000110000000000010110100000000100000110000011000001111000000000001010
- In Low-level code:
  - MOV 45, registerA
  - ADD registerA, 10
- In High-level code:
  - 45 + 10

Compilers vs Interpreters

- There are two types of programs that read high level code and convert it to low-level code:

  1) **Interpreters read the code one line at a time** and execute it on the fly, returning results as the program is being processed.

  2) **Compilers take the entire file and convert it to** low-level code. Then, they hand it off to an executor to actually run it.

Meet Python

- Python is a **high-level language designed for** code readability and simplicity.
- It is an **interpreted language, which means that** we can execute commands one at a time and see the result instantly without compiling.
- It has an extensive library of helpful functions and modules that programmers can use in their code.

Playing with Python

- There are many ways to write and run programs with Python.
- The one we will use is called **Wing and it's an IDE** (Integrated Development Environment).
- Wing IDE 101 is installed on all CDF machines.
- Refer to the course website for detailed installation instructions.

Python

Math

- The simplest way to experiment with the Python shell is to use mathematical operations. Syntax and order of operations is very similar to what we know from math.

Operation	Syntax
Addition	`3 + 5`
Subtraction	`12 - 1`
Multiplication	`3 * 8`
Division	`16 / 4`
Exponentiation	`4 ** 5`

Operators and Operands

- An **operator is a symbol that indicates a simple** operation (i.e. +, -, *, /)
- An **operand is a value on which operations are** performed
- A combination of operators and operands that evaluates to a single value is called an **expression**

Data Types in Python

- Python recognizes and distinguishes between many different types of values.

- The distinction between data types is important because:
  1) different data types take up different amounts of space in memory
  2) different data types support different operations

Data Types in Python

Type	Example	Description
`int`	17	integers between -2147483647 and 2147483647
`long`	3000000000L	integers outside the range above
`float`	3.14159	floating-point decimal e.g. 1.23 or 7.8e-20
`bool`	False	a Boolean ('True' or 'False')
`str`	"Hello!"	a string of characters (text)
`list`	[1.3, 5, "Hi"]	a collection of other values

Data Types in Python

- We can ask Python to give us the type of a particular value:

```
>>> type(25)
<type 'int'>

>>> type(25.0)
<type 'float'>

>>> type(‘Hello World!’)
<type 'str'>
```

Operations With Data Types

- Different data types support different operations.

- Furthermore, certain operations mean different things for different data types:

```
>>> 25 + 15
40
>>> 'Hel' + 'lo'
'Hello'
>>> '25' * 4
'25252525'
```

Operations With Data Types

- When both operands are of the same type, the result of the operation will also be of that type.

- When the operands are different, one of two things will happen:

  1) If they both represent numbers, Python will convert the less precise operand to the type of the other and proceed

  2) In all other cases, Python will raise an error

Quick Word on Errors

- ‘Python will raise an error’ means ‘Python will yell at you when you type in something it can’t understand, sometimes giving you a bit of useful information in the process.’

- There are three types of errors that programmers can make:

1) Syntax Errors

- A syntax error occurs when a program does not conform to the structural and grammatical rules of the programming language.

- In English you can get away with this:

  the meaning of this sentence

  understand you still will

- Python is not as forgiving:

  >>> 4 3 +

  These will be fairly common in your first programs…

2) Runtime Errors

- Runtime errors occur when something goes wrong while the program is running. For instance, the user tries to open a file that doesn't exist or supplies the wrong kind of value.

```
>>> 'Hello' + 9
>>> 45 / 0
```

3) Semantic Errors

- Semantic errors occur when the programmer has given the wrong instructions, so the program does what it was asked, but not what the programmer intended.

  >>> ’25’ + ’15’

- Semantic errors are evil because no language in the world can tell that you've done something wrong.

Operations With Data Types

- When both operands are of the same type, the result of the operation will also be of that type.

- When the operands are different, one of two things will happen:
    1) If they both represent numbers, Python will convert the less precise operand to the type of the other and proceed
    2) In all other cases, Python will raise an error

Revisiting Division

- Given what we know about how Python handles operations, what will be the result of the following:

```
>>> 15 / 3
5
```

- What about this:

```
>>> 17 / 3
5
```

Revisiting Division

- When both operands of a division are int values, Python performs **integer division. The result of** integer division is an integer, and so only the whole quotient is returned.

- However, if at least one of the operands is a float, Python performs floating-point division.

```
>>> 17.0 / 3
5.666666666666667
```

Modulo (%)

- Python has an operator that lets you find the **remainder of an integer division. This operator is** called modulo (%) and looks like this:

```
>>> 17 % 3
2
>>>> 15 % 3
0
```

Variables

- So far, we've used the Python shell to do some simple math, but:

1) we haven't used the results we've obtained for anything

2) we haven't stored anything in memory

Variables

- In computer programming, a **variable is a name** that refers to a value.

- Variables allow programmers to store values, to change them and to use them in later computations.

Assignment Statement

- A line of code that tells Python to do something is called a **statement.**

- In Python, the **assignment statement is used to** assign values to variables. It looks like this:

variable = expression

```
>>> x = 8
>>> x
8
```

Variables as Operands

- A variable can participate in an expression just like any other value.

```
>>> x = 4
>>> 5 + x
9
>>> x = 7
>>> 5 + x
12
```

- As the variable changes, so does the result of the expression.

Variable Types

- In Python, we can assign values of any type to a variable.

```
>>> x = 4
>>> type(x)
<type ‘int’>

>>> x = ‘Hello’
>>> type(x)
<type ‘str’>
```

Variables

- In Python, the assignment operator (=) is used to assign values to variables. It's used in the following format:

variable = expression

1. The expression on the RHS is evaluated and stored in an available memory space

2. The address of that space is assigned to the variable on the LHS

This part is important……..

Computer Science vs Math

- There is a significant difference between what = means in math and in Python.

- E.g. x = 4 + 9
  - In math, it means **equality:**

    "x is always equal to 4 + 9"

  - In Python, it means **assignment:**

    "The variable x now refers to the result of 4 + 9"

Computer Science vs Math

- Equality in math is binding and eternal. If x and y are linked by x = y + 5, every time x changes, y has to change also.

- Assignment in Python is a one-time deal

Computer Science vs Math

- Some things that work in math don't work in Python:

$$4 + 9 = x$$

- Some things that work in Python don't work in math:

$$x = x + 5$$

  - This is the standard way to update a variable's value.

Math Commands

- Python has useful commands for performing calculations.

Command name	Description
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`cos(`***value***`)`	cosine, in radians
`floor(`***value***`)`	rounds down
`log(`***value***`)`	logarithm, base *e*
`log10(`***value***`)`	logarithm, base 10
`max(`***value1***`, `***value2***`)`	larger of two values
`min(`***value1***`, `***value2***`)`	smaller of two values
`round(`***value***`)`	nearest whole number
`sin(`***value***`)`	sine, in radians
`sqrt(`***value***`)`	square root

print Command

- print : produces text output on the console.

- Syntax:

  print "***Message***"

  print ***Expression***

  – Prints the given text message or expression value on the console, and moves the cursor down to the next line.

  print ***Item1***, ***Item2***, ..., ***ItemN***

  – Prints several messages and/or expressions on the same line.

print Command – cont'd

- Examples:

```
print "Hello, world!"
age = 25
print "You have", 65 - age, "years until retirement"
```

Output:

```
Hello, world!
You have 40 years until retirement
```

Input Command

- input : Reads a number from user input.
  - You can assign (store) the result of input into a variable.
  - Example:
    age = input("How old are you? ")
    print "Your age is", age
    print "You have", 65 - age, "years until retirement"

    Output:

    How old are you? **53**
    Your age is 53
    You have 12 years until retirement

E.g. The average of 5 numbers

- **Problem:**

  Given five numbers, find their average.

  - What's the algorithm for solving this problem?
  - And the program?

E.g. The maximum of 5 numbers

- **Problem:**

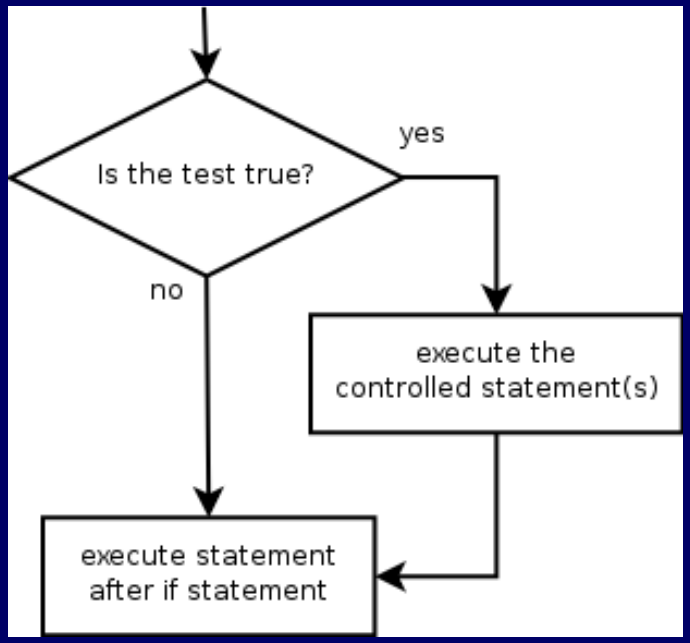
  Given five numbers, find the largest number.

  - What's the algorithm for solving this problem?
  - And the program?

if Statement

- **if statement**: Executes a group of statements only if a certain condition is true. Otherwise, the statements are skipped.

  - Syntax:
    if ***condition***:
        statements



- Example:

```
gpa = 3.4
if gpa > 2.0:
    print "Your application is accepted."
```

Logical operators

- Relational operators results in a True or False result

Operator	Meaning	Example	Result
==	equals	1 + 1 == 2	True
!=	does not equal	3.2 != 2.5	True
<	less than	10 < 5	False
>	greater than	10 > 5	True
<=	less than or equal to	126 <= 100	False
>=	greater than or equal to	5.0 >= 5.0	True

- logical operators

Operator	Example	Result
and	9 != 6 and 2 < 3	True
or	2 == 3 or -1 < 5	True
not	not 7 > 0	False

AND Operator

- Let's look at the relationship between the semantic and logical operator known as the AND operator

- Consider:

  If the *car is fueled AND* the *engine works*,
  then the *engine will start*

AND means that both conditions must be *true* in order for the conclusion to be *true*

OR Operator

- Another basic operator is the OR
- Consider:

  If I have *cash OR a credit card*,
  then *I can pay the bill*

- OR works such that the output is *true*, if either of the two inputs is *true*

if/else Statement

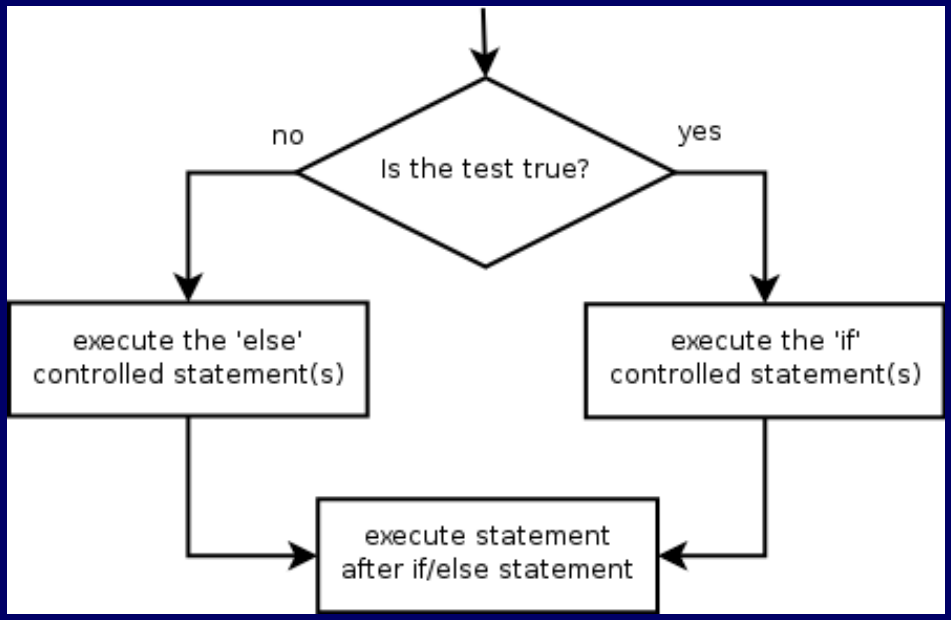
- **if/else statement**: Executes one block of statements if a certain condition is True, and a second block of statements if it is False.
  - Syntax:

    if ***condition***:

        statements

    else:

        statements



- Example:

```
gpa = 1.4
if gpa > 2.0:
    print "Welcome to Mars University!"
else:
    print "Your application is denied."
```

Example

Let the user input 3 numbers

Find if the 3 numbers are the same, in ascending order or descending order.

Also, find if the numbers are all less than 100

What have we learnt today?

- Variables & Data types
- Assignment statement
- Using input and print
- Using math commands (e.g. max)
- if statement
- Logical & Relational operators

This Week's To Do List

- Check out the course website on Blackboard
- Find your CDF username and try to log into a lab computer
- Install Python and Wing IDE (http://www.wingware.com/downloads/wingide-101)
- Register for CodeLab
- Buy textbook