Week 1

Exclusively for Cohort 2
About Us

Sun Jun
Computer Scientist
Uses Windows 8
Buying Lumia 920
Likes Algorithms
Black/White guy

sunjun@sutd
Office: l3r9
About Us

Peter Loh
Part-time Computer Scientist
Uses Windows 7
Paying for son’s iPad
Likes Algorithms and Games
Diablo 3 Enthusiast
About Us

• Huei Yoong
Class Format

not
def Digital World
def Computer

Keyword: single-minded, super-fast, stubborn, etc.
def Programming Language
def Program

- **input:** Get data from the keyboard, a file, or some other device.
- **output:** Display data on the screen or send data to a file or other device.
- **math:** Perform basic mathematical operations like addition and multiplication.
- **conditional execution:** Check for certain conditions and execute the appropriate code.
- **repetition:** Perform some action repeatedly, usually with some variation.
def Programming

THE FRIENDSHIP ALGORITHM
DR. SHELDON COOPER, PH.D.

PLACE PHONE CALL

HOME?
- Yes
  - "WOULD YOU LIKE TO SHARE A MEAL?"
- No
  - LEAVE MESSAGE
    - LEAVE MESSAGE
    - WAIT FOR CALLBACK

WHAT IS THE RESPONSE?
- Yes
  - DINE TOGETHER
    - BEGIN FRIENDSHIP!
- No
  - "DO YOU ENJOY A HOT BEVERAGE?"

WHAT IS THE RESPONSE?
- Yes
  - HAVE TEA
    - CASE: TEA
    - BEGIN FRIENDSHIP!
- No
  - HAVE COFFEE
    - CASE: COFFEE
    - BEGIN FRIENDSHIP!
  - HAVE COCOA
    - CASE: COCOA
    - BEGIN FRIENDSHIP!

RECREATIONAL ACTIVITY?
- Tell me one of your interests?
- No
  - "WHY DON'T WE DO THAT TOGETHER?"
    - BEGIN FRIENDSHIP!
- N>6?
  - Yes
    - CHOOSE LEAST objectionable interest
    - N = N+1
  - No
    - PARTAKE IN INTEREST
    - BEGIN FRIENDSHIP!
<table>
<thead>
<tr>
<th>Description</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time to Destination</td>
<td>23:56</td>
</tr>
<tr>
<td>Estimated Time of Arrival</td>
<td>15:12</td>
</tr>
<tr>
<td>Local Time at Destination</td>
<td>15:16</td>
</tr>
<tr>
<td>Local Time at Origin</td>
<td>15:16</td>
</tr>
<tr>
<td>Distance to Destination</td>
<td>30 km</td>
</tr>
</tbody>
</table>
def Debugging
Debugging syndrome

• “the computer hates me”
• “the program only works when I wear my hat backward”
• Random walk programming: “if I try if-while-for-recursion all of them, one must work!”
• “ahhhhhhhhh”
def Debugging

• “debugging is like detective work. You are confronted with clues, and you have to infer the processes and events that led to the results you see.”

• “When you have eliminated the impossible, whatever remains, however improbable, must be the truth.” - A. Conan Doyle, The Sign of Four
def Debugging

• Syntax error: boring

• Runtime error: slightly more interesting

• Semantic error: 0.1+0.2 is not 0.3?
Python as Calculator

With some slides from Hans Petter Langtangen
Binary Numbers

• Computers are binary.
  – decimal number to a binary
  – 2’s Complement in N-bits
    • \(-x\) is \(2^N - x\)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Unsigned value</th>
<th>2’s complement value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01111111</td>
<td>127</td>
<td>127</td>
</tr>
<tr>
<td>01111110</td>
<td>126</td>
<td>126</td>
</tr>
<tr>
<td>00000010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>00000001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>00000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11111111</td>
<td>255</td>
<td>-1</td>
</tr>
<tr>
<td>11111110</td>
<td>254</td>
<td>-2</td>
</tr>
<tr>
<td>10000010</td>
<td>130</td>
<td>-126</td>
</tr>
<tr>
<td>10000001</td>
<td>129</td>
<td>-127</td>
</tr>
<tr>
<td>10000000</td>
<td>128</td>
<td>-128</td>
</tr>
</tbody>
</table>

8-bit two's-complement integers
Binary Numbers

• Binary operators
  – AND: a & b
  – OR: a | b
  – XOR: a^b
  – Complement: ~a
  – Shift: a <<2, a >> 2

• Example: “0b1011”
  – Get binary string: bin(111)
  – Get octal string and hexadecimal string: oct(), hex()
Exercise

• Tutor: 192.168.2.136/digitalworldtutor/tutor2/

• Problem Wk.1.2.1: Integer to Binary
We will learn programming through examples

The first examples involve programming of formulas

Here is a formula for the position of a ball in vertical motion, starting at $y = 0$ at time $t = 0$:

$$y(t) = v_0 t - \frac{1}{2}gt^2$$

- $y$ is the height (position) as function of time $t$
- $v_0$ is the initial velocity (at $t = 0$)
- $g$ is the acceleration of gravity
- Computational task: given $v_0$, $g$ and $t$, compute $y$
What is a program?

A sequence of instructions to the computer, written in a programming language, which is somewhat like English, but very much simpler – and very much much stricter!

In this course we shall use the Python language

Our first example program:

Evaluate $y(t) = v_0 t - \frac{1}{2}gt^2$ for $v_0 = 5$, $g = 9.81$ and $t = 0.6$:

$$y = 5 \cdot 0.6 - \frac{1}{2} \cdot 9.81 \cdot 0.6^2$$

Python program for doing this calculation:

```python
print 5*0.6 - 0.5*9.81*0.6**2
```
How to write and run the program

- A (Python) program is plain text
- First we need to write the text in a *plain text editor*
- Use Gedit, Emacs or IDLE *(not MS Word or OpenOffice!)*
- Write the program line
  
  ```python
  print 5*0.6 - 0.5*9.81*0.6**2
  ```
- Save the program to a file (say) `ball_numbers.py`
  (Python programs are (usually) stored files ending with `.py`)
- Go to a terminal window
- Go to the folder containing the program (text file)
- Give this operating system command:
  ```shell
  Unix/DOS> python ball_numbers.py
  ```
- The program prints out 1.2342 in the terminal window
Exercise: Hello World

• Execute this in the shell/prompt
>>> print ‘Hello World’

• Save this in a file and execute it
>>> press F5 in IDLE
When you use a computer, you always run a program.

The computer cannot do anything without being precisely told what to do, and humans write and use programs to tell the computer what to do.

Some anticipate that programming in the future may be as important as reading and writing (!).

Most people are used to double-clicking on a symbol to run a program – in this course we give commands in a terminal window because that is more efficient if you work intensively with programming.

In this course we probably use computers differently from what you are used to.
Would you consider these two lines to be "equal"?

- `print 5*0.6 - 0.5*9.81*0.6**2`
- `write 5*0.6 - 0.5*9.81*0.6^2`

Humans will say "yes", computers "no"

The second line has no meaning as a Python program.
`write` is not a legal Python word in this context, and the hat does not imply $0.6^2$

We have to be extremely accurate with how we write computer programs!

It takes time and experience to learn this.

"People only become computer programmers if they’re obsessive about details, crave power over machines, and can bear to be told day after day exactly how stupid they are." – G. J. E. Rawlins
Would you consider these two lines to be "equal"?
print 5*0.6 - 0.5*9.81*0.6**2
write 5*0.6 - 0.5*9.81*0.6^2

Humans will say "yes", computers "no"

The second line has no meaning as a Python program

write is not a legal Python word in this context, and the hat does not imply $0.6^2$

We have to be extremely accurate with how we write computer programs!

It takes time and experience to learn this

“People only become computer programmers if they’re obsessive about details, crave power over machines, and can bear to be told day after day exactly how stupid they are.” –G. J. E. Rawlins
Would you consider these two lines to be "equal"?

\[
\text{print } 5*0.6 - 0.5*9.81*0.6^{**2} \\
\text{write } 5*0.6 - 0.5*9.81*0.6^{^2}
\]

Humans will say "yes", computers "no"

The second line has no meaning as a Python program.

\textit{write} is not a legal Python word in this context, and the hat does not imply 0.6^2.

We have to be extremely accurate with how we write computer programs!

It takes time and experience to learn this.

"People only become computer programmers if they’re obsessive about details, crave power over machines, and can bear to be told day after day exactly how stupid they are." – G. J. E. Rawlins
Storing numbers in variables

- From mathematics you are used to variables, e.g.,

\[ v_0 = 5, \quad g = 9.81, \quad t = 0.6, \quad y = v_0 t - \frac{1}{2}gt^2 \]

- We can use variables in a program too, and this makes the last program easier to read and understand:

```python
v0 = 5
g = 9.81
t = 0.6
y = v0*t - 0.5*g*t**2
print y
```

- This program spans several lines of text and use variables, otherwise the program performs the same calculations and gives the same output as the previous program
- In mathematics we usually use one letter for a variable.
- In a program it is smart to use one-letter symbols, words or abbreviation of words as names of variables.
- The name of a variable can contain the letters a-z, A-Z, underscore _, and the digits 0-9, but the name cannot start with a digit.
- Variable names are case-sensitive (e.g., a is different from A).
- Example on other variable names in our last program:
  ```python
  initial_velocity = 5
  accel_of_gravity = 9.81
  TIME = 0.6
  VerticalPositionOfBall = initial_velocity*TIME - \n    0.5*accel_of_gravity*TIME**2
  print VerticalPositionOfBall
  ```
  (the backslash allows an instruction to be continued on the next line)
- Good variable names make a program easier to understand!
Some words are reserved in Python

- Certain words have a special meaning in Python and cannot be used as variable names.
- These are: `and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `exec`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `not`, `or`, `pass`, `print`, `raise`, `return`, `try`, `with`, `while`, `and`, `yield`.
- There are many rules about programming and Python, we learn them as we go along with examples.
Comments are useful to explain how you think in programs.

Program with comments:

```python
# program for computing the height of a ball
# in vertical motion
v0 = 5  # initial velocity
g = 9.81  # acceleration of gravity
t = 0.6  # time
y = v0*t - 0.5*g*t**2  # vertical position
print y
```

- Everything after `#` on a line is ignored by the computer and is known as a comment where we can write whatever we want.
- Comments are used to explain what the computer instructions mean, what variables mean, how the programmer reasoned when she wrote the program, etc.
Comments are not always ignored....

- Normal rule: Python programs, including comments, can only contain characters from the English alphabet

- Norwegian characters,
  
  ```python
  hilsen = 'Kjære Åsmund!' # er æ og Å lov i en streng?
  print hilsen
  ```
  
  will normally lead to an error:
  
  ```python
  SyntaxError: Non-ASCII character ...
  ```

- Remedy: put this line as the first line in your program:
  
  ```python
  # -*- coding: latin-1 -*-
  ```
  
  (this special comment line is not ignored - Python reads it...)

- Another remedy: stick to English everywhere in a program
Given $C$ as a temperature in Celsius degrees, compute the corresponding Fahrenheit degrees $F$:

$$F = \frac{9}{5}C + 32$$

Program:

```python
C = 21
F = (9/5)*C + 32
print F
```

Execution:

```
Unix/DOS> python c2f_v1.py
53
```

We must always check that a new program calculates the right answer(s): a calculator gives 69.8, not 53

Where is the error?
Integer division

- 9/5 is not 1.8 but 1 in most computer languages (!)
- If \(a\) and \(b\) are integers, \(a/b\) implies integer division: the largest integer \(c\) such that \(cb \leq a\)
- Examples: \(1/5 = 0\), \(2/5 = 0\), \(7/5 = 1\), \(12/5 = 2\)
- In mathematics, 9/5 is a real number (1.8) – this is called float division in Python and is the division we want
- One of the operands (\(a\) or \(b\)) in \(a/b\) must be a real number ("float") to get float division
- A float in Python has a dot (or decimals): \(9.0\) or \(9.\) is float
- No dot implies integer: \(9\) is an integer
- \(9.0/5\) yields 1.8, \(9/5.\) yields 1.8, \(9/5\) yields 1
- Corrected program (with correct output 69.8):
  ```
  C = 21
  F = (9.0/5)*C + 32
  print F
  ```
Example: $\frac{5}{9} + 2a^4/2$, in Python written as $5/9 + 2*a**4/2$

The rules are the same as in mathematics: proceed term by term (additions/subtractions) from the left, compute powers first, then multiplication and division, in each term

- $r_1 = 5/9 \ (=0)$
- $r_2 = a**4$
- $r_3 = 2*r_2$
- $r_4 = r_3/2$
- $r_5 = r_1 + r_4$

Use parenthesis to override these default rules – or use parenthesis to explicitly tell how the rules work (smart):

$$(5/9) + (2*(a**4))/2$$
Precedence

\[ a,b,c,d,e = 20,10,15,5,0 \]
\[ e = (a + b) \times c / d \]
\[ e = ((a + b) \times c) / d \]
\[ e = (a + b) \times (c / d) \]
\[ e = a + (b \times c) / d \]

Homework:
- Find out the binding power of all the operators – and forget them
Strings

• Example: “sunjun”
• String Concatenation: ‘sun’ + ‘jun’
• String to Integer: int(“35”)
• rjust
  – The method \texttt{rjust()} returns the string right justified in a string of length \textit{width}.
Type Conversion

• Every data object has a type: `type()`
• Built-in functions for type conversion
  – `int()`
  – `str()`
  – `float()`
Input

• x = raw_input(“message”)
  – x is a string

• y = input(“message”)
  – y is the result of evaluating the input
Exercise

Write a Python program to convert F to C, taking F as an input from Console: \( C = (5/9) \times (F - 32) \)
"printf-style" formatting of text and numbers

- Output from calculations often contain text and numbers, e.g. 
  At t=0.6 s, y is 1.23 m.

- We want to control the formatting of numbers 
  (no of decimals, style: 0.6 vs 6E-01 or 6.0e-01)

- So-called *printf formatting* is useful for this purpose:

  ```
  print 'At t=%g s, y is %.2f m.' % (t, y)
  ```

- The printf format has "slots" where the variables listed at the end are put: 
  %g ← t, %.2f ← y
Examples on different printf formats

- \%g: most compact formatting of a real number
- \%f: decimal notation (-34.674)
- \%10.3f: decimal notation, 3 decimals, field width 10
- \%.3f: decimal notation, 3 decimals, minimum width
- \%e or \%E: scientific notation (1.42e-02 or 1.42E-02)
- \%9.2e: scientific notation, 2 decimals, field width 9
- \%d: integer
- \%5d: integer in a field of width 5 characters
- \%s: string (text)
- \%-20s: string, field width 20, left-adjusted

See the book for more explanation and overview
Triple-quoted strings ("""") can be used for multi-line output, and here we combine such a string with printf formatting:

```python
v0 = 5
g = 9.81
t = 0.6
y = v0*t - 0.5*g*t**2

print """
At t=%f s, a ball with
initial velocity v0=%.3E m/s
is located at the height %.2f m.
"""
% (t, v0, y)
```

Running the program:
```
Unix/DOS> python ball_output2.py

At t=0.600000 s, a ball with
initial velocity v0=5.000E+00 m/s
is located at the height 1.23 m.
```
Output: Escape Characters

• Print ‘\’
  – \\n  – ‘
  – ”

Homework:
• Find out all the escape characters – and know where to find the info later
Exercise

• Tutor:

http://192.168.2.136/digitalworldtutor/tutor2/
  – Problem Wk.1.2.2: Compute Ball Height
  – Problem Wk.1.2.3: Compute Ball Height 2
a, b, c = 4, -7, 10
p1 = a < b and b < c
print p1
p2 = a < b or b < c
print p2
p3 = not a < b
print p3
print a > b or b > c and not a < c
print (a > b or b < c) and not a < c
print a > b or (b < c and not a < c)
What if we need to compute \( \sin x \), \( \cos x \), \( \ln x \), etc. in a program?

Such functions are available in Python’s `math` module.

In general: lots of useful functionality in Python is available in modules – but modules must be imported in our programs.

Compute \( \sqrt{2} \) using the `sqrt` function in the `math` module:

```python
import math
r = math.sqrt(2)
# or
from math import sqrt
r = sqrt(2)
# or
from math import *  # import everything in math
r = sqrt(2)
```

Another example:

```python
from math import sin, cos, log
x = 1.2
print sin(x)*cos(x) + 4*log(x)  # log is ln (base e)
```
Exercise

• Tutor:
  http://192.168.2.136/digitalworldtutor/tutor2/
  – Problem Wk.1.2.4: Time to Reach Height
Some frequently used computer science terms

- Program or code or application
- Source code (program text)
- Code/program snippet
- Execute or run a program
- Algorithm (recipe for a program)
- Implementation (writing the program)
- Verification (does the program work correctly?)
- Bugs (errors) and debugging

Computer science meaning of terms is often different from the natural/human language meaning
Some frequently used computer science terms

- Program or code or application
- Source code (program text)
- Code/program snippet
- Execute or run a program
- Algorithm (recipe for a program)
- Implementation (writing the program)
- Verification (does the program work correctly?)
- Bugs (errors) and debugging

Computer science meaning of terms is often different from the natural/human language meaning.
Some frequently used computer science terms

- Program or code or application
- Source code (program text)
- Code/program snippet
- Execute or run a program
- Algorithm (recipe for a program)
- Implementation (writing the program)
- Verification (does the program work correctly?)
- Bugs (errors) and debugging

Computer science meaning of terms is often different from the natural/human language meaning
A program consists of statements

```
    a = 1    # 1st statement
    b = 2    # 2nd statement
    c = a + b # 3rd statement
    print c  # 4th statement
```

- Normal rule: one statement per line
- Multiple statements per line is possible with a semicolon in between the statements:
  ```
  a = 1; b = 2; c = a + b; print c
  ```

- This is a print statement:
  ```
  print 'y=%g' % y
  ```

- This is an assignment statement:
  ```
  v0 = 3
  ```

Assignment: evaluate right-hand side, assign to left-hand side

```
    myvar = 10
    myvar = 3*myvar     # = 30
```
Syntax

- Programs must have correct syntax, i.e., correct use of the computer language grammar rules, and no misprints.

- This is a program with two syntax errors:

  ```
  myvar = 5.2
  prinnt Myvar
  ```

  (`prinnt` is an unknown instruction, `Myvar` is a non-existing variable)

- Python reports syntax errors:

  ```
  prinnt Myvar
  ```

  `SyntaxError: invalid syntax`

- Only the first encountered error is reported and the program is stopped (correct error and continue with next error).

"Programming demands significantly higher standard of accuracy. Things don’t simply have to make sense to another human being, they must make sense to a computer." – Donald Knuth
• Programs must have correct syntax, i.e., correct use of the computer language grammar rules, and no misprints

• This is a program with two syntax errors:

  \[
  \text{myvar} = 5.2 \\
  \text{prinnt Myvar}
  \]

  (\text{prinnt} \text{ is an unknown instruction, Myvar is a non-existing variable})

• Python reports syntax errors:

  \[
  \text{prinnt Myvar}
  \]

  \text{SyntaxError: invalid syntax}

• Only the first encountered error is reported and the program is stopped (correct error and continue with next error)

"Programming demands significantly higher standard of accuracy. Things don’t simply have to make sense to another human being, they must make sense to a computer." – Donald Knuth
Programs must have correct syntax, i.e., correct use of the computer language grammar rules, and no misprints.

This is a program with two syntax errors:

```python
myvar = 5.2
printn Myvar
```

(printn is an unknown instruction, Myvar is a non-existing variable)

Python reports syntax errors:

```python
printn Myvar
```

SyntaxError: invalid syntax

Only the first encountered error is reported and the program is stopped (correct error and continue with next error)

“Programming demands significantly higher standard of accuracy. Things don’t simply have to make sense to another human being, they must make sense to a computer.” – Donald Knuth
Blanks may or may not be important in Python programs.

These statements are equivalent (blanks do not matter):

\[
\begin{align*}
v0 &= 3 \\
v0 &= 3 \\
v0 &= 3 \\
v0 &= 3 \\
v0 &= 3
\end{align*}
\]

(the last is the preferred formatting style of assignments)

Here blanks do matter:

```python
while counter <= 10:
    counter = counter + 1  # correct (4 leading blanks)

while counter <= 10:
    counter = counter + 1  # invalid syntax
```

(more about this in Ch. 2)
A program has some known input data and computes some (on beforehand unknown) output data

Sample program:

```python
v0 = 3; g = 9.81; t = 0.6
position = v0*t - 0.5*g*t*t
velocity = v0 - g*t
print 'position:', position, 'velocity:', velocity
```

Input: $v_0$, $g$, and $t$

Output: position and velocity
An operating system (OS) is a set of programs managing hardware and software resources on a computer.

Example:

```
Unix/DOS> emacs myprog.py
```

`emacs` is a program that needs help from the OS to find the file `myprog.py` on the computer’s disk.

- Linux, Unix (Ubuntu, RedHat, Suse, Solaris)
- Windows (95, 98, NT, ME, 2000, XP, Vista)
- Macintosh (old Mac OS, Mac OS X)
- Mac OS X ≈ Unix ≈ Linux ≠ Windows

Python supports cross-platform programming, i.e., a program is independent of which OS we run the program on.
Objects

- Everything in Python is an object
- Variables refer to objects
  
  ```
  a = 5    # a refers to an integer (int) object
  b = 9    # b refers to an integer (int) object
  c = 9.0  # c refers to a real number (float) object
  d = b/a  # d refers to an int/int => int object
  e = c/a  # e refers to float/int => float object
  s = 'b/a=%g' % (b/a)  # s is a string/text (str) object
  ```

- We can convert between object types:
  
  ```
  a = 3    # a is int
  b = float(a)  # b is float 3.0
  c = 3.9  # c is float
  d = int(c)  # d is int 3
  d = round(c)  # d is float 4.0
  d = int(round(c))  # d is int 4
  d = str(c)  # d is str '3.9'
  e = '-4.2'  # e is str
  f = float(e)  # f is float -4.2
  ```
Complex Numbers

• Unified treatment of complex and real functions
  – from math import sqrt
  – from cmath import sqrt
  – from numpy.lib.scimath import *
Exercise

• Tutor:
  – Part 1.2: Cohort Session 2, 5a-f
Let us compute $\frac{1}{49} \cdot 49$ and $\frac{1}{51} \cdot 51$:

```python
v1 = 1/49.0*49
v2 = 1/51.0*51
print '%.16f %.16f' % (v1, v2)
```

Output with 16 decimals becomes

```
0.9999999999999999 1.0000000000000000
```

Most real numbers are represented inexactly on a computer.

Neither $\frac{1}{49}$ nor $\frac{1}{51}$ is represented exactly, the error is typically $10^{-16}$.

Sometimes such small errors propagate to the final answer, sometimes not, and sometimes the small errors accumulate through many mathematical operations.

Lesson learned: real numbers on a computer and the results of mathematical computations are only approximate.
Another example involving math functions

The sinh $x$ function is defined as

$$\sinh(x) = \frac{1}{2} (e^x - e^{-x})$$

We can evaluate this function in three ways:

1) $	ext{math.sinh}$,
2) combination of two $	ext{math.exp}$,
3) combination of two powers of $	ext{math.e}$

```python
from math import sinh, exp, e, pi
x = 2*pi
r1 = sinh(x)
r2 = 0.5*(exp(x) - exp(-x))
r3 = 0.5*(e**x - e**(-x))
print '%.16f %.16f %.16f' % (r1,r2,r3)
```

Output: $r_1$ is 267.7448940410164369, $r_2$ is 267.7448940410164369, $r_3$ is 267.7448940410163232 (!)
Round-off Errors

• How to store 0.1 in memory?
• $0.1 + 0.2 = 0.30000000000000004$
• $\text{round}(2.675, 2) = 2.67$
  – 2.67499999999999982236431605997495353221893310546875
Random Numbers

• import random
• import random.random()
• Import random.randint()

There is no real random number!
Exercise

• Tutor:
  – Part 1.2: Cohort Session 2, 6
Programs must be accurate!

- Variables are names for objects
- We have met different object types: int, float, str
- Choose variable names close to the mathematical symbols in the problem being solved
- Arithmetic operations in Python: term by term (+/-) from left to right, power before * and / – as in mathematics; use parenthesis when there is any doubt
- Watch out for unintended integer division!
Mathematical functions like \( \sin x \) and \( \ln x \) must be imported from the `math` module:

```python
from math import sin, log
x = 5
r = sin(3*log(10*x))
```

Use `printf` syntax for full control of output of text and numbers

```python
>>> a = 5.0; b = -5.0; c = 1.9856; d = 33
>>> print 'a is', a, 'b is', b, 'c and d are', c, d
a is 5.0 b is -5.0 c and d are 1.9856 33
```

Important terms: object, variable, algorithm, statement, assignment, implementation, verification, debugging
Alan Perlis, computer scientist, 1922-1990:

- "You think you know when you can learn, are more sure when you can write, even more when you can teach, but certain when you can program”
- "Within a computer, natural language is unnatural”
- "To understand a program you must become both the machine and the program”
Exercise

• Tutor:
  – Part 1.2: Cohort Session 2, 7-9