TDT4127 Programming and Numerics Week 35

Programming basics and floating point numbers

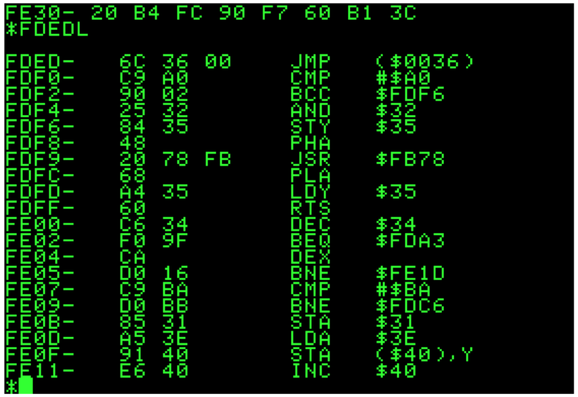
Learning goals

- Goals
  - Learn about programming
  - Learn about using Python
  - Learn about data types

- Curriculum
  - Starting out with Python, ch. 1

What is programming?

- Telling the computer ***exactly*** what to do
  - We use a *programming language* to give instructions
  - **Python** is one of many programming languages
  - Programming language gets *compiled* into machine code
    - Luckily, this process is invisible to us
  - The machine code basically boils down to large amounts of:
    - «Read this number»
    - «Add these numbers»
    - «Multiply these numbers»
    - «Check if a > b»
    - «Switch to this instruction»



What is a program?

- A set of *instructions* telling the computer what to do.
  - *Declare variables* and *assign values* to them
    - As in mathematics: «Let *x* = 5. Let *y* = 6».
  - *Do calculations* with variables
    - As in mathematics: «Let *z* = x + y»
  - Make branching decisions
    - More of this than in mathematics: «If z > 10, do <something>»
  - Other *high-level* instructions
    - Plot graphs, show video, look for user input, etc.
- Programs *execute* line by line, just like we read recipes.
  - What's written first, happens first.

How to install and use Python

- Download Python here
  - Get the version for your operating system
  - Choose standard installation
- You can use IDLE (now installed) to write/run programs
  - Two ways to do this
    - «Live» programming in the shell, like using a calculator
      - Mostly useful for testing functionality, data is lost when IDLE is closed
    - Scripted programming: write code in a separate file, then execute
      - Standard way of writing programs, saving code for later on
  - See the introductory exercise or do a Python tutorial online
- Python is a nice programming language
  - Hides some of the gritty details behind readable code
  - Easier to focus on the actual functionality instead of details

Programming versus mathematics

- Computer: Physical, all information is stored digitally (on/off states of transistors/capacitors) as 0's and 1's
  - Space limitation! We can only have so many units to store data.

    $\pi = 3.14159265359$
- Mathematics: Information is abstract, represented symbolically
  - No space limitation! We can speak of infinitely large/small quantities.

    $\pi = \pi$

Programming versus mathematics

- Mathematics (or physics)
  - Equations express truths, e.g.
    - $(a + b)^2 = a^2 + 2ab + b^2$
    - $E = mc^2$
  - The equality symbol = means the left side *is equal to* right side
- Programming:

  Statements are imperative sentences, giving orders:

  = means assignment,

  `x = 3;` «let x be 3»

  `x = x + 1;` «calculate 3+1, let x now be 4»

  (this would be meaningless as a mathematical equation)

  == means comparison, `x == y`

  if `x` and `y` have the same value, == will calculate to True, otherwise False

  In both cases we instruct the computer to **do** something

  (=) remembering a value in a variable,

  (==) make a comparison, conclude with True or False

Data types

- This is quite hidden when using Python, but a computer has different ways of representing data.
  - All a computer stores is 0s and 1s
    - Organized such that it knows what the 0s and 1s represents
  - Strings (words or letters) are one *data type*
  - Numbers are split in three data types:
    - **Integers**: Whole numbers
    - **Floating point numbers (floats)**: Real numbers
    - *Complex numbers* (not used in this course)
  - Integers and floats have different representations and uses

Different types of numbers

- Computers are limited
  - By the number of transistors in their processing units
  - By the number of bytes of storage available
- Numbers are unlimited – in different ways
  - Integers (0,1,-1,2,-2,3…) are *countably* infinite and can be infinitely large
  - Real numbers (all decimal numbers) are *uncountably* infinite; between numbers *a* and *b* there are infinitely many more
    - Nowhere to start counting! (0.001 is smaller than 0.01 < 0.1 etc.)
- We need limited representations of unlimited numbers
  - Depending on what kind of number it is

Integers

- We write our whole numbers (integers) in base 10:
    - 3145 = 5*1 + 4*10 + 1*100 + 3*1000
    - The *k*'th spot represents multiples of 10^{k-1}
- Computers are constructed in terms of bits (on/off switches) and most naturally use base 2:
    - 100101 = 1*1 + 0*2 + 1*4 + 0*8 + 0*16 + 1*32
    - The *k*'th spot represents multiples of 2^{k-1}
    - 100101 is a *6 bit number*
- Representations of integers are *exact*

Integers

- For two computers make the same sense of a number, we need standards.
  - A standard long signed integer (Python default) has 32 bits
    - 1 bit to assign negative/positive values
    - 31 bits to represent number value
    - Ex: -131 = 00000000000000000000000010000011
    - Largest value $1*2^0 + 1*2^1 \ldots + 1*2^{30} = 2*2^{30} - 1 = 2\ 147\ 483\ 647$
  - Python can represent «infinitely» large integers
    - This happens «behind the scenes», we don't need to take care

Floating point numbers

- Decimal numbers can be both infinitely *large* and *long*
  - For example, π is infinitely long
    - $\pi = 3.14159265359\ldots$
  - We can still use it mathematically:
    - $A = \pi r^2$
  - When computing, we use a *truncated* value with an uncertainty:
    - $\pi = 3.14\ (\pm\ 0.005)$
  - We do this for other infinitely long numbers as well:
    - $1/3 = 0.3333\ (\pm\ 0.005)$
- Our representation of decimal numbers must balance *magnitude* and decimal point *precision.*

Floating point numbers

- Floats are a tradeoff between *size range* and *accuracy*
- Based on scientific notation for numbers
  - Avogadro's number: $10^{23} \times 6.022140857$
  - Electron rest mass: $10^{-31} \times 9.109383561$
  - Large range of numbers, here using only 12 digits (base 10 numbers).
  - *Uncertainty* lies in the last digit
- Floating point numbers use the same idea, but in base 2
  - $a = (-1)^{sg} \times 2^{e-b} \times s$
    - Sign: *sg* is 1 bit representing 0 or 1, allows negative/postive numbers
    - Exponent: *e* is a positive integer, adjusts size
    - Bias: *b* is a *predetermined* integer allowing for negative exponents
    - Significand: s is a number between 1 and 2 of the form

$$s = 1.s_1 s_2 s_3 s_4 s_5 s_6 \ldots$$
$$= 1 + s_1 \times 2^{-1} + s_2 \times 2^{-2} + s_3 \times 2^{-3} + s_4 \times 2^{-4} + s_5 \times 2^{-5} + s_6 \times 2^{-6} + \ldots$$

- This is like scientific notation in base 2, with uncertainty in the last digit.
- More in Exercise 1, after which we will mostly not have to worry about them.

Operations with floating point numbers

- Addition/subtraction **requires care** due to roundoff error
  - To add two numbers we match their exponents, so the smaller number loses significance
  - Example in base 10: 12345.67 + 1.224567 with 7 digit precision:

$$
\begin{array}{ll}
 & 12345.67 \\
+ & 1.224567 \\
\hline
= & 12346.894567 \approx 12346.89
\end{array}
$$

  - Same effect as adding 1.22 since the last four digits are lost.
  - When adding a large amount of small numbers to a larger number, we lose precision unless it is done carefully.
    - Kahan's algorithm is an algorithm for doing so. Not curriculum.
    - Other workarounds exist.
      - Not standard due to the extra computation time needed.

Operations with floating point numbers

- Multiplication/division are **safe**
  - We add/subtract exponents and multiply/divide the significands.
- Checking for equality is **very unsafe**
  - If *a* and *b* are floats, $a = b$ if all their bits are the same.
  - Due to imprecision, numbers that *should* be equal after some computation, may not be equal.
  - Example: Are $d = (a + b) + c$ and $e = a + (b + c)$ equal?

$a = 123456.7$, $b = 123.4567$, $c = 0.4567891$

$d = 123580.2 + 0.4567891 = 123580.7$

$e = 123456.7 + 123.9135 = 123580.6$

NTNU

Information about exercises

- Special teaching assistants have been assigned for numerics questions
  - Sitting at drop-in lab in A3-107 at Realfagbygget at given hours:
    - Mondays 10:00 – 16:00
    - Tuesdays 10:00 – 16:00
    - Wednesdays 12:00 – 16:00
    - Thursdays 10:00 – 12:00
    - Fridays 10:00 – 16:00
  - Programming questions can be asked at any computer lab.

Summary

- Install Python, get started on programming!
- Computers operate with different types of numbers
- Integers are used for whole numbers and are **exact**
- Floating point numbers are used for real (decimal) numbers and are **inexact**
- Addition of small and large numbers can cause problems
- Do not make code that relies on checking whether two floats are equal
  - Integers, on the other hand, are okay!

Reference group

- Have three already, need one or two more
- Around 4 hours of work for a nice line on your CV

  ...You also get to influence the course!
- Will meet a few times after class (3-5 minutes) and 1-2 times outside of class during the semester.

Questions?