Computers and Programs

Theresa Migler-VonDollen
CMPS 5P

Computers and Programs

- A modern computer can be defined as “a machine that stores and manipulates information under the control of a changeable program.”
- Two elements:
    - Computers are devices for manipulating information.
    - Computers operate under the control of a changeable program.

Computers and Programs

- What is a computer program?
  - A detailed, step-by-step set of instructions telling a computer what to do.
  - If we change the program, the computer performs a different set of actions or a different task.
  - The machine stays the same, but the program changes.
  - Programs are executed, or carried out.

Programming

- All computers have the same power, with suitable programming, i.e. each computer can do the things any other computer can do.
- Software (programs) rule the hardware (the physical machine).
- The process of creating this software is called programming.

Programming

- Why learn to program?
  - Fundamental part of computer science
  - Having an understanding of programming helps you have an understanding of the strengths and limitations of computers.
  - Helps you become a more intelligent user of computers
  - It can be fun.
  - Helps the development of problem solving skills, especially in analyzing complex systems by reducing them to interactions between simpler systems.
  - Programmers are in great demand.

What is Computer Science?

- It is NOT the study of computers.
  - “Computers are to computer science what telescopes are to astronomy.” - E. Dijkstra
- The question is: “What can be computed?”

What is Computer Science?

- Design
  - One way to show a particular problem can be solved is to actually design a solution.
  - This is done by developing an *algorithm*, a step-by-step process for achieving the desired result.
  - One problem - it can only answer the question of what can be computed in the positive. You can't prove a negative.

What is Computer Science?

- Analysis
  - Analysis is the process of examining algorithms and problems mathematically.
  - Some seemingly simple problems are not solvable by any algorithm. These problems are said to be unsolvable.
  - Problems can be intractable if they would take too long or take too much memory to be of practical value.

What is Computer Science?

- Experimentation
  - Some problems are too complex for analysis.
  - Implement a system and then study its behavior.

Hardware Basics

The central processing unit (CPU) is the “brain” of a computer.

- The CPU carries out all the basic operations on the data.
- Examples: simple arithmetic operations, testing to see if two numbers are equal.

Hardware Basics

Memory stores programs and data.

- CPU can only directly access information stored in main memory (RAM or Random Access Memory).
- Main memory is fast, but volatile, i.e. when the power is interrupted, the contents of memory are lost.
- Secondary memory provides more permanent storage: magnetic (hard drive, floppy), optical (CD, DVD)

Hardware Basics

Input devices

- Information is passed to the computer through keyboards, mice, etc.

Output devices

- Processed information is presented to the user through the monitor, printer, etc..

Hardware Basics

Fetch-Execute cycle

- First instruction retrieved from memory
- Decode the instruction to see what it represents
- Appropriate action carried out.
- Next instruction fetched, decoded, and executed.
- Repeat.

Programming Languages

Natural language has ambiguity and imprecision problems when used to describe complex algorithms.

- Programs expressed in an unambiguous, precise way using programming languages.
- Every structure in programming language has a precise form, called its syntax.
- Every structure in programming language has a precise meaning, called its semantics.

Programming Languages

A programming language is like a code for writing the instructions that the computer will follow.

- Programmers will often refer to their program as computer code.
- Process of writing an algorithm in a programming language often called coding.

Programming Languages

High-level computer languages

- Designed to be used and understood by humans

Low-level computer languages

- Computer hardware can only understand a very low level language known as machine language

Programming Languages

Add two numbers - Low-level

- Load the number from memory location 2001 into the CPU
- Load the number from memory location 2002 into the CPU
- Add the two numbers in the CPU
- Store the result into location 2003

In reality, these low-level instructions are represented in binary (1′s and 0′s)

Binary numbers

- The modern binary number system was discovered by Gottfried Leibniz in 1679.
- Counting in binary:
  - 0 - 0
  - 1 - 1
  - 2 - 10
  - 3 - 11
  - 4 - 100
  - 5 - 101
- What is 137 in binary?

Programming Languages

Add two numbers -High-level

- $c = a + b$
- This needs to be translated into machine language that the computer can execute.
- Compilers convert programs written in a high-level language into the machine language of some computer.

Programming Languages

- Interpreters simulate a computer that understands a high-level language.
- The source program is not translated into machine language all at once.
- An interpreter analyzes and executes the source code instruction by instruction.

Programming Languages

Compiling vs Interpreting

- Once program is compiled, it can be executed over and over without the source code or compiler. If it is interpreted, the source code and interpreter are needed each time the program runs.
- Compiled programs generally run faster since the translation of the source code happens only once.

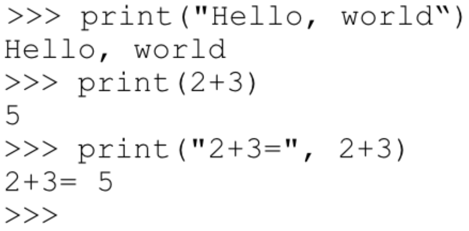
Programming Languages

Compiling vs Interpreting

- Interpreted languages are part of a more flexible programming environment since they can be developed and run interactively
- Interpreted programs are more portable, meaning the executable code produced from a compiler for a Pentium won't run on a Mac, without recompiling. If a suitable interpreter already exists, the interpreted code can be run with no modifications.

Python

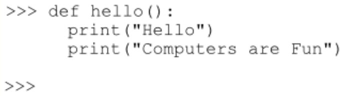
When you start Python, you will see something like >>>
>>> is called a *prompt* indicating that Python is ready for us to give it a command. These commands are called statements.



```
>>> print("Hello, world“)
Hello, world
>>> print(2+3)
5
>>> print("2+3=", 2+3)
2+3= 5
>>>
```

Python

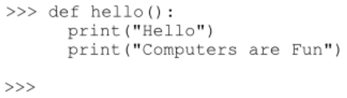
Usually we want to execute several statements together that solve a common problem. Use a function.



```
>>> def hello():
        print("Hello")
        print("Computers are Fun")

>>>
```

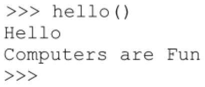
- The first line tells Python we are defining a new function called hello.
- The following lines are indented to show that they are part of the hello function.
- The blank line (hit enter twice) lets Python know the definition is finished.

Python



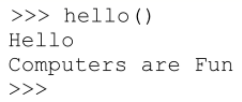
```
>>> def hello():
        print("Hello")
        print("Computers are Fun")

>>>
```

- Notice that nothing has happened yet! We’ve defined the function, but we haven’t told Python to perform the function.
- A function is invoked by typing its name.



```
>>> hello()
Hello
Computers are Fun
>>>
```

Python



```
>>> hello()
Hello
Computers are Fun
>>>
```

- What are the “( )” for?
- Commands can have changeable parts called parameters that are placed between the ( )’s

Practice with Parameters

Write a Python function that takes in a number as a parameter and prints the square of that number.

Practice with Parameters

Write a Python function that takes in two numbers as parameters and prints the sum of the squares of the two numbers.

Python Programs

- When we exit the Python prompt, the functions we've defined cease to exist.
- Programs are usually composed of functions, modules, or scripts that are saved on disk so that they can be used again and again.
- A module file is a text file created in text editing software (saved as "plain text") that contains function definitions.
- A programming environment is designed to help programmers write programs and usually includes automatic indenting, highlighting, etc.

Python Programs

- We'll use filename.py when we save our work to indicate it's a Python program.
- In this code we're defining a new function called main.
- The main() at the end tells Python to run the code.

Practice with a Python program

Write a Python program that greets the user, and asks for an input number, call it x. Do the following 10 times:
$x = 3.9 * x * (1 - x)$.
Call this program chaos.py

Python comments

- Lines that start with # are called comments
- Intended for human readers and ignored by Python
- Python skips text from # to end of line

Python main function

```
def main():
```

- Beginning of the definition of a function called main
- Since our program has only this one module, it could have been written without the main function.
- The use of main is customary, however.

Python variables

```
x = eval(input("Enter a number between 0 and 1: "))
```

- *x* is an example of a *variable*
- A variable is used to assign a name to a value so that we can refer to it later.
- The quoted information is displayed, and the number typed in response is stored in x.

Python loops

```
for i in range(10):
```

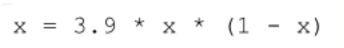
- For is a *loop* construct
- A loop tells Python to repeat the same thing over and over.
- In this example, the following code will be repeated 10 times.

Python body

```
x = 3.9 * x * (1 - x)
print(x)
```

- These lines are the *body* of the loop.
- The body of the loop is what gets repeated each time through the loop.
- The body of the loop is identified through indentation.
- The effect of the loop is the same as repeating this two lines 10 times.

Python assignment



```
x = 3.9 * x * (1 - x)
```

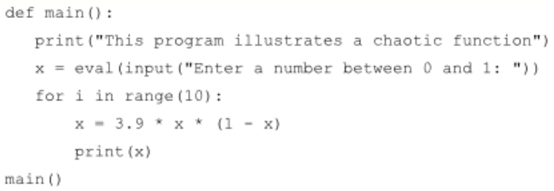
- This is called an *assignment* statement
- The part on the right-hand side of the $=$ is a mathematical expression.
- $\star$ is used to indicate multiplication
- Once the value on the right-hand side is computed, it is stored back into (assigned) into x

Python execute main

```
main()
```

- This last line tells Python to execute the code in the function main

Python chaos.py program



```
def main():
   print("This program illustrates a chaotic function")
   x = eval(input("Enter a number between 0 and 1: "))
   for i in range(10):
       x = 3.9 * x * (1 - x)
       print(x)
main()
```

- For any given input, returns 10 seemingly random numbers between 0 and 1.
- It appears that the value of *x* is chaotic....