Introduction to Computer Science and to Python

CS 8: Introduction to Computer Science, Spring 2019
Lecture #2

Ziad Matni, Ph.D.
Dept. of Computer Science, UCSB

Your Instructor

Your instructor: **Ziad Matni, Ph.D** *(zee-ahd mat-knee)*

Email: ***zmatni@cs.ucsb.edu***

(please put **CS8** at the start of the subject header)

My office hours:
Mondays 1:00 PM – 3:00 PM, at **SMSS 4409**

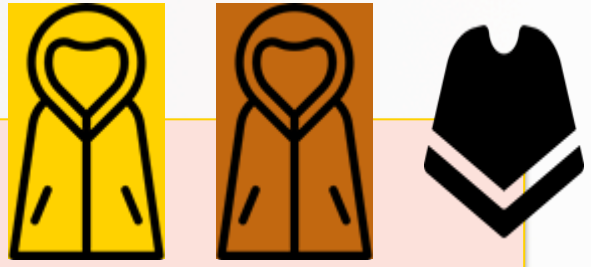
A Word About Registration for CS8

- This class is **FULL**,

  & the waitlist is **CLOSED**.

Examples of Everyday Use of Algorithms

- **Problem to Solve**: What coat, if any, should I wear today?
- **Algorithm**:



1. Measure the outdoor temperature, T.
2. If T < 62F then wear my blue coat.
    1. If blue coat is *dirty* (dirt level ≥ 7), wear my brown coat instead
    2. If it's also *raining* (Now raining = True), wear my black poncho instead
3. If T ≥ 62F then don't wear a coat
    1. Plan on buying ice-cream for lunch!

And Now, With More Detail…

1. Measure the outdoor temperature, T.
2. If T < 62F then wear my blue coat.
    1. If blue coat is *dirty* (dirt level ≥ 7), wear my brown coat instead
    2. If it's also *raining* (Now raining = True), wear my black poncho instead
3. If T ≥ 62F then don't wear a coat
    1. Plan on buying ice-cream for lunch!

a) Define outcomes:

1. wear blue coat,
2. wear brown coat,
3. wear black poncho,
4. wear nothing and get ice-cream for lunch!

b) Define conditions:

1. T < 62 or not
2. Dirt_Level < 7 or not
3. Now_Rain = True or not

b) Get measures/values for T, Dirt_Level, Now_Rain

c) If ((T < 62) AND (Dirt_Level < 7)) then (outcome = 1)
d) If ((T < 62) AND (Dirt_Level ≥ 7)) then (outcome = 2)
e) If ((T < 62) AND (Now_Rain = True)) then (outcome = 3)
Otherwise (outcome = 4)
f) The End

And Now, With "Language"…

1. Measure the outdoor temperature, T.
2. If T < 62F then wear my blue coat.
    1. If blue coat is *dirty* (dirt level ≥ 7), wear my brown coat instead
    2. If it's also *raining* (Now raining = True), wear my black poncho instead
3. If T ≥ 62F then don't wear a coat
    1. Plan on buying ice-cream for lunch!

```
Measure(T)
Get(Dirt_Level)
Assess(Now_Raining)

if (T < 62) AND (Dirt_Level < 7)
    then Outcome = 1
if (T < 62) AND (Dirt_Level >= 7)
    then Outcome = 2
if (T < 62) AND (Now_Raining = True)
    then Outcome = 3
else
    Outcome = 4

End Program
```

…that has specific form and syntax

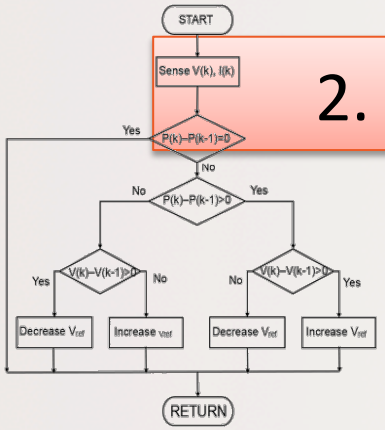
(like any "language" would!)

This is often called "pseudo-code" and is the pre-cursor to writing a program in a specific computer language

What is “Computer Science”?

The study of :

1. The designs and uses of computers as useful ***tools*** in our daily lives



2. The use of algorithms to solve problems

mostly around the creation, processing, interpreting, communication, etc…
of information

Computer Systems

- **Hardware**
  - The physical computer
    - CPU, Memory ICs, Printed circuit boards
    - Plastic housing, cables, etc…
- **Software**
  - The instructions and the data fed to/generated by the computer
    - Programs and applications
    - Operating systems

What is Programming?

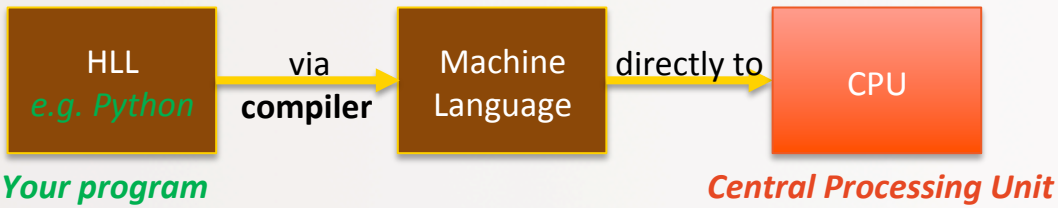
Instructing a computer what to do

- Programs – a.k.a. "Software"
  - Includes operating system, utilities, applications, …
  - Computer just sits there until instructions fed to CPU

- **Machine language** – basic CPU instructions
  - Completely numeric (as binary numbers) – i.e., computer "readable"
  - Specific to particular computer types – not portable

High-Level Computer Languages

- **A way to program computers using "human-like" language**
  - Easier to write/read (than 1s and 0s…):
    - `e.g.` ***`result = (first + second)`*** *`instead of`* ***`"1001111010101010110"`***
  - Translated to machine language by ***compiler programs***
    - Advantage: the same H-LL Program can be used on different machines!



High-Level Language Paradigms

- Procedural languages – focus is on *functions and process*
- Early languages include FORTRAN, PASCAL, BASIC, C
- More modern "object-oriented" languages – focus on *objects (more on those later)*
  - Includes **C++** and **Java**
- Multi-paradigm languages – has combined features
  - e.g., **Python** (invented in 1991… and still evolving)

~1991…2019…

- First developed as a language designed for *learning how to progra*
  - Guido van Rossum →
- Python is Open Sourced technology since it's first version (1991)
  - So it is free!
  - Has a huge community of volunteer developers
  - Guido still involved
- Lots of handy modules ready to use at **http://docs.python.org/**
  - More on modules later…

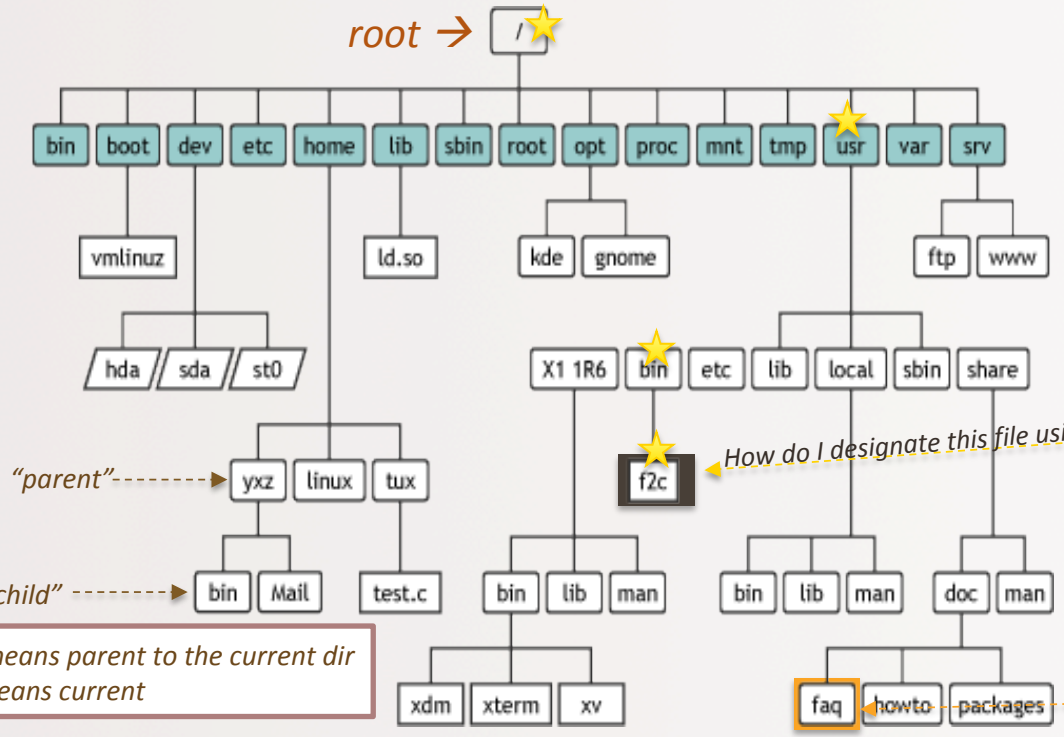
Linux

- Is an operating system
  - Like MacOS or Windows
- Operating systems are the largest pieces of software on a computer
  - They help all the other pieces of software run seamlessly with inputs and outputs
- We will be using Linux in our CSIL labs to run our Python programs

A Brief Intro to Linux

- In Linux, we mostly interact with the OS by **typing out** our commands
  - As opposed to, say, using a point-and-click menu
- I will introduce you to a few of these commands that are most commonly used to manage files and folders
- The ***$*** symbol indicates a Linux prompt

Organization of Files in a Computer



Is done **hierarchically**
Uses **folders** (aka directories)
Starts at the "**root**" directory
designated with a /

Basic Linux Commands

cd = change directory

```
$ cd MyDirectory ↵
```
change directory to MyDirectory

```
$ cd ↵
```
go to my "home" directory

pwd = present working directory

```
$ pwd ↵
```
show me what directory I am in

Yellow Band = Class Demonstration! ☺

Basic Linux Commands

ls = list files

```
$ ls
```

show me all the files in my directory

cp = copy files

```
$ cp file1.txt file2.txt
```

*copy **file1.txt** into **file2.txt** (arbitrary names)*
(the order matters)

mv = move (rename) files

```
$ mv file1.txt file2.txt
```

*rename **file1.txt** as **file2.txt***
(the order matters)

Basic Linux Commands

cat = concatenate file

```
$ cat file1.txt
```

*show me what's in **file1.txt***

more = show me more of the file

```
$ more file1.txt
```

show me what's in file1.txt, but one screen's worth at a time (good for long files)

Python IDLE

- *IDLE* is what we use to demonstrate Python in class
  - You can also use it at home (download info given last class)

- If you want to create a *Python program*, then you will place *all* the program code inside a text file
  - Text file always ends in **.py**
  - You can *create* and also *run (execute)* the **.py** program from Python IDLE

- Make sure the version of IDLE you use is **AT LEAST 3.7.*x* or LATER**

1st Python Lesson!

- Numbers and Arithmetic in Python
- Variables in Python
- Variable Types in Python
- Operations in Python
- Assignment versus Comparison of Values

Yellow Band = Class Demonstration! ☺

Numbers are Objects to Python

- Each object *type* has: data and related operations
- 2 basic number types
  - **Integers** (like `5` or `-72`) – add, subtract, multiply, …
  - **Floating point** numbers (like `0.005` or `-7.2`) – operations similar but *not exactly the same as integer* operations
- Expect many ***non-number object*** types later in the quarter…
  - But they also have data and related operations

Common Data Types

Type	Example	Description
float	3.1415	A **real** number. Can be positive or negative.

Common Data Types

Type	Example	Description
float	3.1415	A **real** number. Can be positive or negative.
int	3	An **integer** number. Can be positive or negative.
str	“ILUVCS8!!!” “Gaucho Goop”	A series (or a **string**) of characters. Note the use of “ ” as delimiters.
bool	True False	A **Boolean** outcome of a **logical comparison**. *(Note: True, False have upper-case first letters)*

Arithmetic Operators

`+ - * /` add, subtract, multiply, (ordinary) divide

`%` modulus operator – remainder

`( )` means whatever is inside is evaluated first

`**` raise to the power

What's easier to remember:
*3 * 2 – 1*
Or:
*(3 * 2) – 1 ???*

Special Python division operator for integers:

`//` result is truncated: `7 // 2` gives me `3` (not 3.5)

Precedence rules:

1. `()`
2. `**`
3. `*, /, %, //`
4. `+, -`
5. `=`

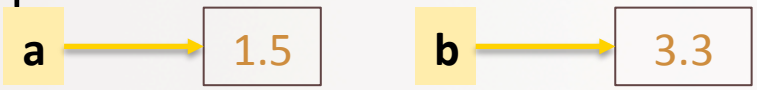
Comments in Python

- Anything placed after the # symbol is considered a "comment"
  - Is completely ignored by the compiler
  - Typically place commentary next to code for the benefit of others (humans) reading our code

Variables

- A **variable** is a *symbolic* reference to data

- The variable's **name** represents *what* information it contains



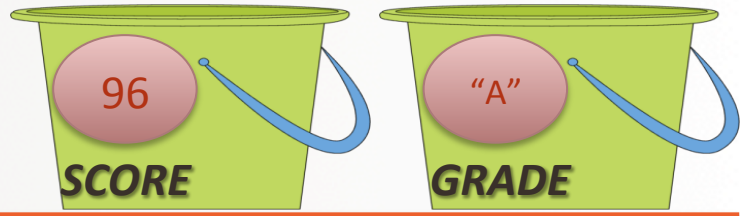
- They are called "**variables**" because

  --- data can VARY or change ---

  while **operations** on the variable remain the same

  – e.g. Variables "a" and "b" can take on different *values*, but I may always want to add them together

Variables



- Variables are like "buckets" that can keep data
  - You can label these buckets with a **name**
  - When you reference a bucket, you use its name, not the data stored in the bucket
  - You can "re-use" the buckets

- If two variables are of the same ***type***, you can perform ***operations*** on them

Variables in Python

- We assign a **value** to variables with the *assignment operator* `=`
  - Example: `a = 3`
- We can change that value stored
  - Example: `a = 5  # it's not 3 any more!!!`

Assigning Names to Variables

- Variable names are actually references
- Like "pointers" to objects
- Can have multiple references to the same object

```
x = 5     # x refers to an integer
y = x     # Now x and y refer to the same object
```

Assigning Names to Variables

- Dynamic typing is a key Python feature
- Any legal name can point to any ***data type*** – even different types at different times

```
x = 5       # x refers to an integer
y = x       # Now x and y refer to the same object
            # and both are integers
x = 1.2     # Now x refers to floating point 1.2
            # (y still refers to the integer 5)
```

All Data in Python Has a *type()*

- But you can change its type
  - *Implicitly*, like in the last slide
  - *Explicitly*, by forcing the type
- Introducing the built-in function *type()*
- Let's try these out on IDLE (note: the **>>>** is just the prompt)

```
>>> grade = 3.8
>>> type( grade )
>>> grade = 5
>>> type( grade )
>>> type("Green Eggs")
>>> type(True)
>>> type(true)
```

```
>>> pi = 3.14
>>> type( pi )
>>> p = int( pi )
>>> print( p )
>>> type ( p )
```

YOUR TO-DOs

- ❑ Read **Chapter 2**
- ❑ Finish **Homework1** (due **Tuesday!**)
- ❑ Prepare for **Lab1** next week (Monday!)

- ❑ Hug a tree! *But don't get wet…*

</LECTURE>