3. More NLP; system and programming basics LING 471

Logistics

- Office hour is drop-in
  - Come for any question!
- UW Undergraduate Linguistics Society
  - Weekly meetings on Thursdays from 6pm to 7pm in Savery Hall 168
  - Linktree at https://linktr.ee/ulsuw
  - Instagram page @uls.uw
  - Email ulsuw@uw.edu

Pointers about readings

- Important for Thursday
  - Some papers have very technical parts, and even experts don't understand an entire paper all at once
  - You should identify some reasonable goals for yourself when reading and try to meet those goals
- Documentation
  - Sometimes, the assignment includes documentation sites
  - This can be boring but is often necessary to fully understand what you are doing

Assignment 1

- Due April 14
- Please start now!
  - You can do some parts already
  - Skip the steps you can't do
  - Finished fast? Great!
  - Don't put yourself in a position where you need to do an impossible amount hours before it's due (esp. if you'll need to ask questions)
- Several parts: 1 is data focused, the rest are setup focused

Learning outcomes

- Discuss what evaluation of NLP systems is
- Identify and list basic facts about common operating systems
- Define high-level and low-level w.r.t. systems and programs
- Describe basic differences between procedural/imperative and declarative/functional languages
- Describe basic differences between compiled and interpreted languages
- Describe basic differences between static and dynamic typing
- Write a program that prints strings in Python

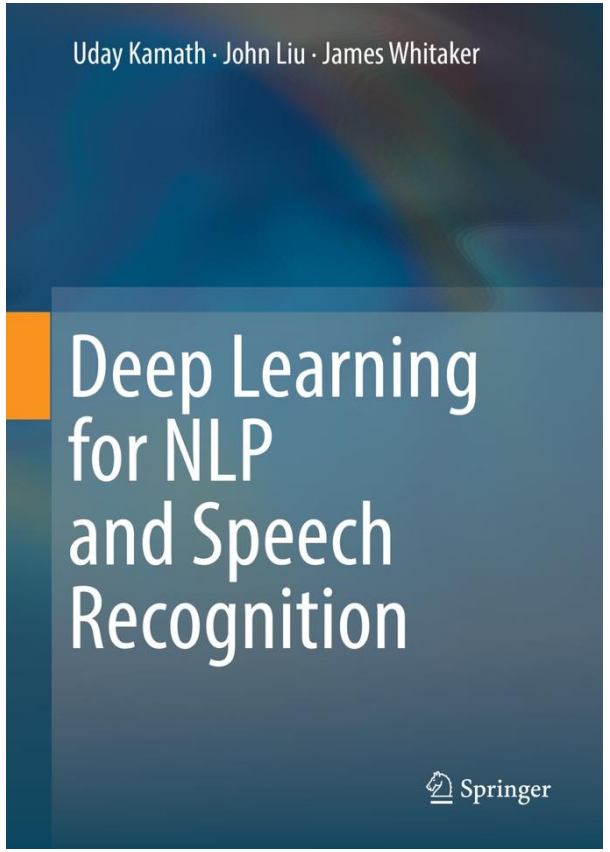
A bit more on NLP

NLP and machine learning

- Machine learning (ML) studies computer algorithms that are automatically improved by getting feedback
  - Correct prediction? Reinforce!
  - Wrong prediction? Adjust!
- Correct and incorrect are labels
  - ML doesn't know what these labels mean
  - "Correct" = 1, "incorrect" = 0
- Sometimes, labels can be automatically obtained

Is NLP a machine learning or deep learning subdiscipline?

- Today, probably so!
- 10 years ago?
  - Not so clearly
  - Other methodologies were used
- 10 years from now?
  - We should train a model and see what it forecasts…

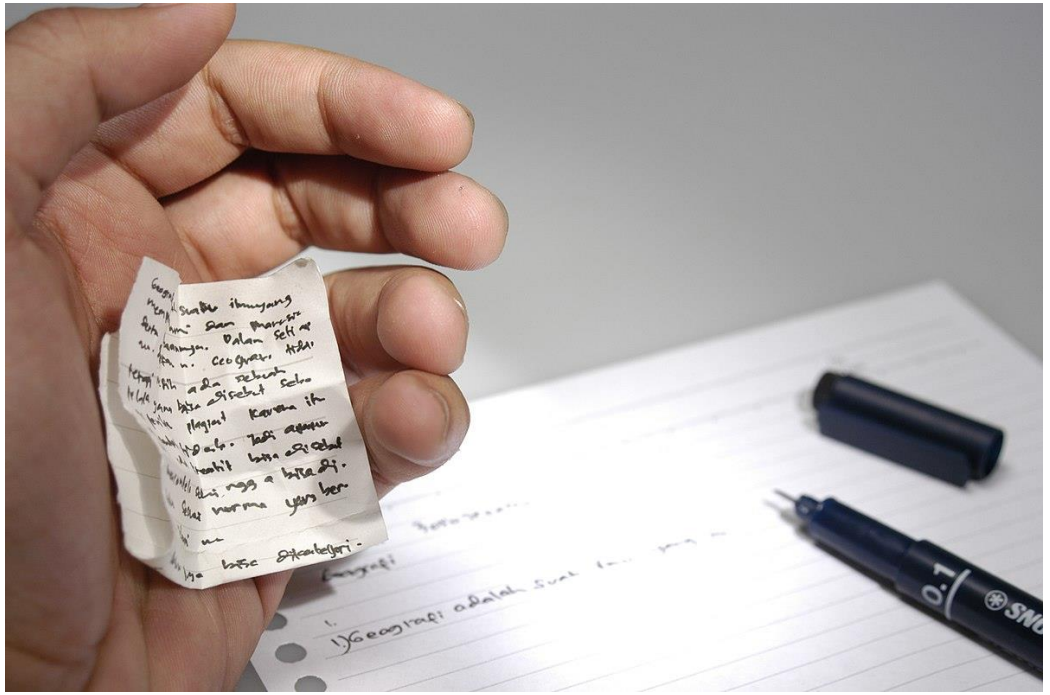


NLP and evaluation

- NLP is shaped by **evaluation**
  - Computing **metrics** that indicate how well the system performed on held-out data
  - E.g., accuracy, precision/recall, coverage, etc.
- Held out data: Data not previously seen by system **or** developer

NLP evaluation vs. training

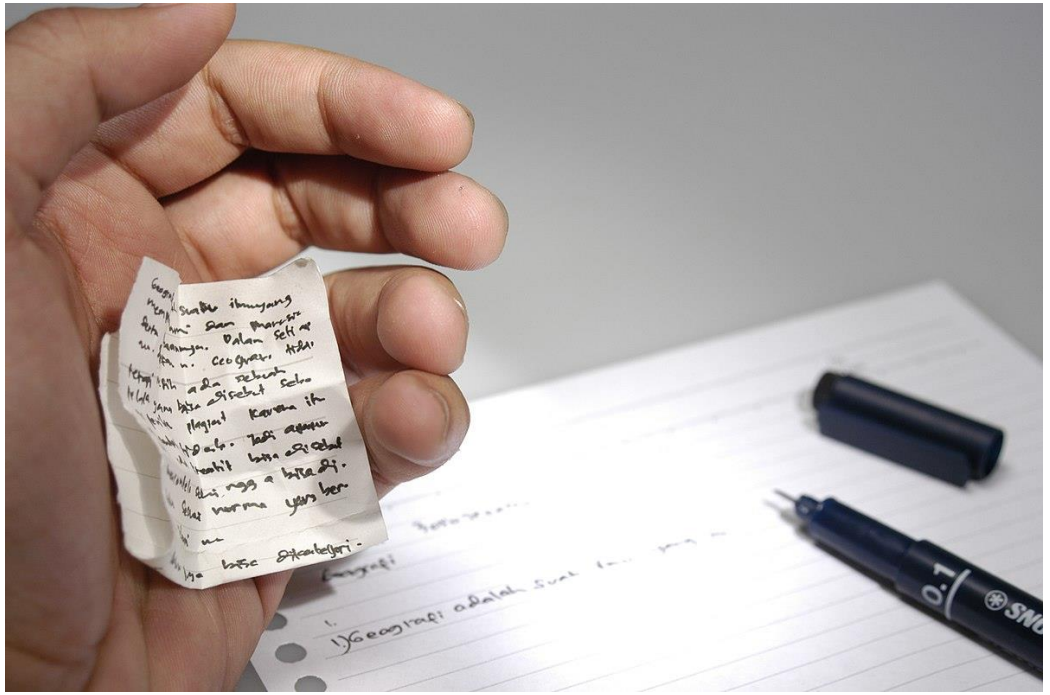
- During training, the algorithms are adjusted
- How about during evaluation? Should adjustments be made?



https://commons.wikimedia.org/wiki/File:Cheating.JPG
CC BY-SA 3.0, from Hariadhi

NLP evaluation vs. training

- During training, the algorithms are adjusted
- How about during evaluation? Should adjustments be made?
  - They should not
  - You are otherwise “cheating” by looking at data you’re being tested on



https://commons.wikimedia.org/wiki/File:Cheating.JPG
CC BY-SA 3.0, from Hariadhi

NLP evaluation

- To evaluate systems
  - **Test** data must be **labeled**
    - Depending on type of ML used for training
- Labels can be created by humans or be inherent
  - What exactly is a non-human created label?
  - If non-human-created label is **interpreted** by a human, what does that mean?
- Most “unsupervised” or “fully automatic” NLP systems **rely on humans** because the systems must be **evaluated**

Operating system basics

Why do we need programming?

- Humans like to conserve energy
  - Doing things by hand is tedious (some might say soul-crushing)
- Humans are comparatively slow
  - Processing LOTS of data manually is impossible
  - Making time-aligned phonetic transcriptions can be 400x real time
- Humans make mistakes
  - We often complete tasks inconsistently and our attention is not predictable
- Automation “solves” these problems

What is an operating system?

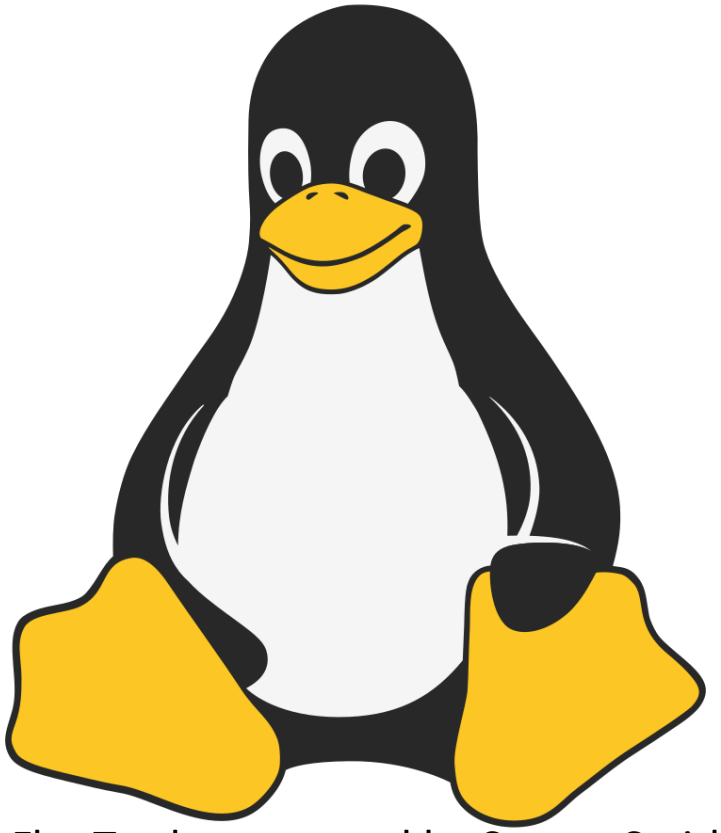
- Operating systems are software that support basic functions
  - E.g., how does at program get executed at the hardware level?
  - Memory allocation, file locations, I/O
- No program can run without an OS

Operating systems are everywhere!

- Every computer has some kind of operating system
- Examples: Windows, macOS, iOS, Android, Unix, etc.
  - There are a lot more!
  - Many small appliances have custom OSs

Levels of understanding

- Understanding an OS requires understanding **low-level** software
  - Less abstraction over hardware
- We will be programming using **high-level** language
  - More abstraction over hardware
- Why learn about system?



Flat Tux logo created by Sergey Smith based on original by Larry Erwing

Systems and configurations

- **Some** programming is possible to do in the web browser
  - **High-level** programming
  - Why bother learning **low-level** stuff?
- Data science projects are **pipelines** and work as a **system**
- Putting modules together is one of the **hardest** things in programming

Landscape of operating systems

OLDER

- Unix (AT&T's Bell labs)
- MS-DOS (Microsoft)

NEWER

- Linux (open source, Linux **community**)
  - Unix-**like**
- macOS (Apple)
  - Unix-**based**
- Windows (Microsoft)
  - DOS-based
- Cloud
  - Someone else's computer

What does it mean to be based on Unix or DOS, or Unix-like?

- Unix-like and Unix-based systems are common in research and engineering
  - Linux and BSD are free
  - Considered more flexible, stable, and secure
- DOS-based Windows is common elsewhere
  - Established market
  - Considered more user-friendly
    - (But is this just a factor of familiarity?)

Practical differences for Unix/Unix-like systems

- (I will start calling these *nix systems for brevity)
- Command-line language is different
- Different text file format
- Different file path separator (uses “/” instead of “\”)
  - Technical note: “/” is forward-slash or slash, “\” is backward-slash or backslash
- Different applications available

Practical OS differences for us?

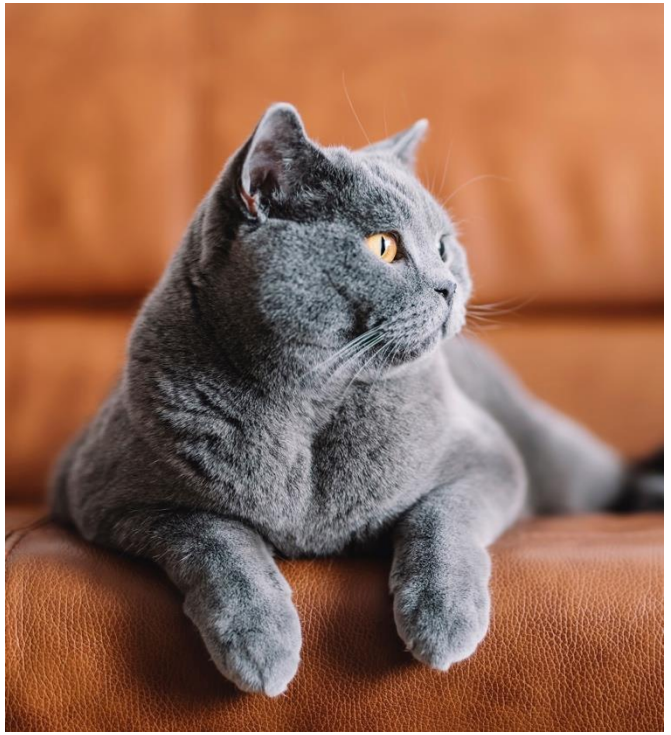
- Not much, really
- When I demo something, it's mostly on MacOS
  - You should be able to replicate demos on your system
  - Sometimes you will need to Google "How do I do X on Windows/Mac/Linux/…?"
- You may need to do some things on a remote Linux cluster

A digression into linguistics: Conceptual metaphor

- We'll continue on OSs in a moment, but we need a concept from linguistics first
- **(Conceptual) metaphor**: Understanding one thing in terms of another
  - Often understanding an **abstract** concept in terms of a **concrete** one (but not always)
- Example: TIME IS MONEY
  - You can *spend time*, *invest time*, *lose time*
- Prevalent throughout language use and logical reasoning

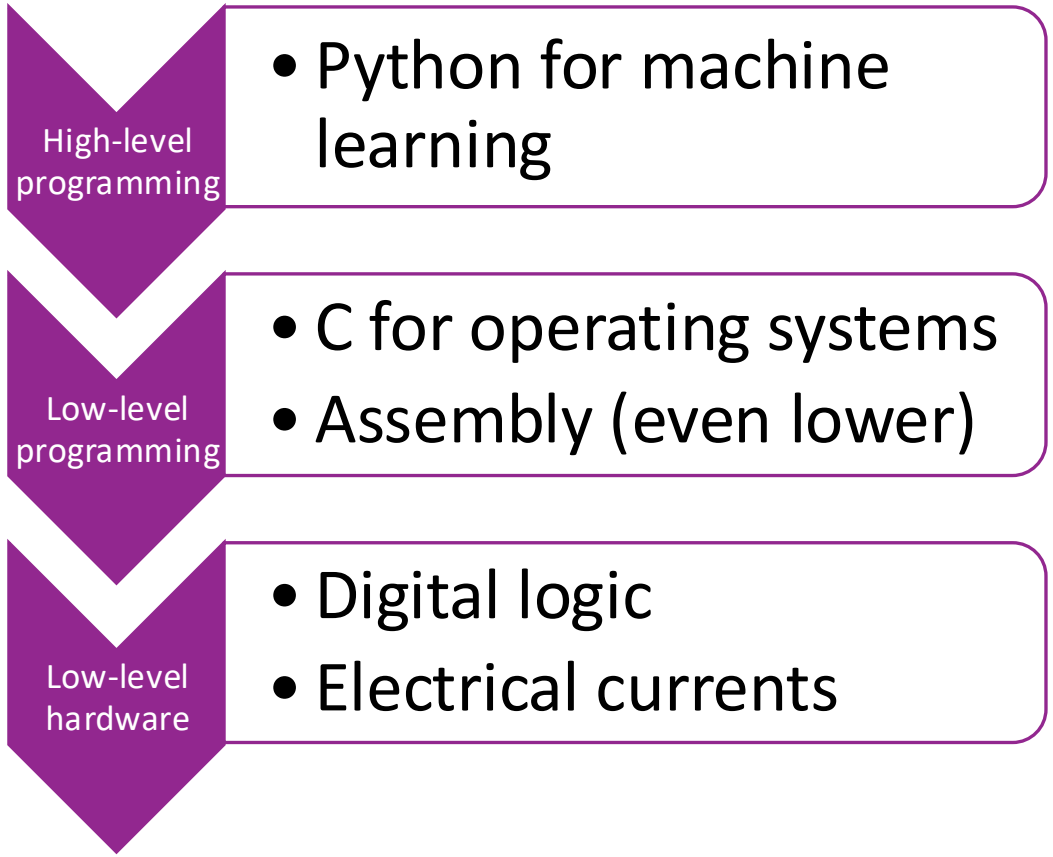
High-level and low-level

- We've talked about high-level and low-level a lot. But, what are they?
- Low-level is basic information, often too detailed to analyze effectively
  - E.g., every pixel in an image, every frame in a video
- High-level is more abstract information, often summarizing low-level information
  - E.g., image of a cat, video on cooking



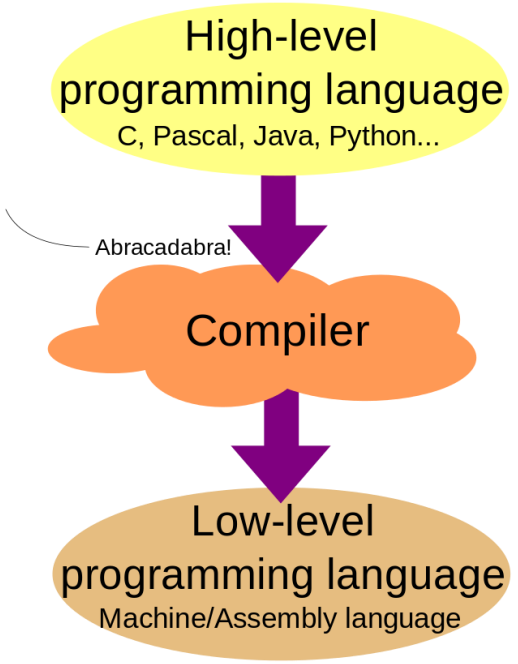
High-level and low-level for computers

- In totality, hardware is the lowest-level (electrical currents in transistors)
  - Too much information to think about complex programs this way
- Assembly is, roughly, lowest software level
- Each level higher builds and **abstracts** over lower level
  - Importantly, these abstractions can (eventually) be transformed into lower levels (reverse is harder)



Programming and levels

- A high-level program is written as text
  - Conforms to syntactic rules and uses valid keywords
- In order for program to be executed
  - Interpreter or compiler will translate it to lower-level instructions
  - Hardware can take these instructions and set transistors on/off
- Reductionistically, we reason about transistor states through the high-level program
  - **Conceptual metaphor**!



https://commons.wikimedia.org/wiki/File:High_level_to_low_level_diagram.svg
CC BY-SA 3.0 from Kayau

Piecing together high-level programs

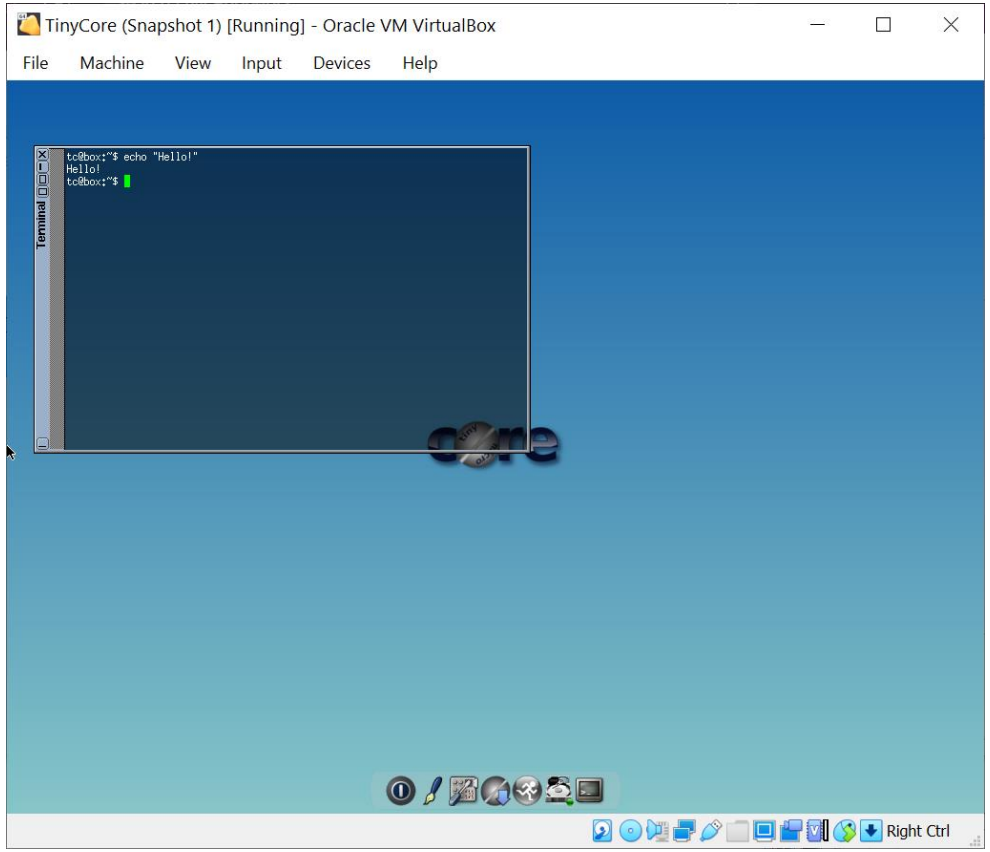
- A high-level program consists of parts
  - Pre-build libraries, modules, and your code
- Without pre-built libraries and modules, you would need to write everything from scratch
  - Including the compiler or interpreter
  - You (usually) don't want to do this at any large scale for applied tasks like NLP
- The computer needs to know **exactly** where all these parts are

Juggling different OSs

- Different physical machines or hard drives
  - Can be expensive
- Virtual machines
  - Run within an extant operating system using software
  - Good for small-to-medium tasks and casual gaming
- Remote machines (like **patas**)
  - Can be necessary to access powerful hardware you otherwise can't afford

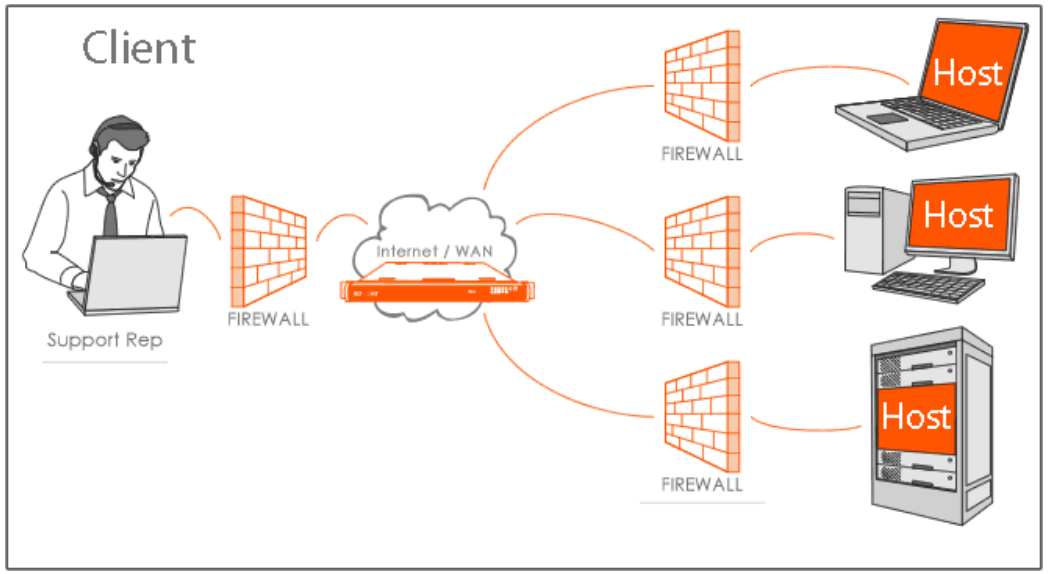
Virtual machines

- Your host OS dedicates some space on the disk
  - Where it installs a guest OS
  - Software examples: VirtualBox (free), VMWare ($)
- You probably won't need one of these in class, but you should be aware that they exist
  - You need hard drive space for them



Remote servers

- You can connect to a computer over the internet
- We may not use them much in this class, but if you continue on in data science/NLP, you will end up using them a lot



https://commons.wikimedia.org/wiki/File:Client_versus_Host_network_diagram_example_-_Remote_desktop.gif
CC BY-SA 4.0 from Ludopedia

Programming language basics

Programming languages

- We will be using Python
- Other languages you may have heard of: C/C++, C#, Java, JavaScript, Ruby, Haskell, Go, R, MATLAB, Julia, Objective-C, COBOL, Fortran…
- A particular programming language can be more or less suited for a task
  - (Not true of human/natural languages!)
- Some languages are more popular than others, regardless of how well suited they are for a given task

What exactly is a programming language?

- At a rather abstract level, it is a combination of **tokens** and **rules** about how those tokens can be combined
  - Together, these form a **grammar**
  - Roughly, this corresponds to morphology (word structure) and syntax (sentence structure) in linguistics
- Tokens examples: keywords (like “for” and “if”), numbers (3.1415), strings (“Hello!”)
- Syntax
  - How tokens can be arranged into larger units and in what order
    - Is `3+2` valid? What about `+ 3 2`? `)print(“Hello!` ? `3 = x + 2 = y`?

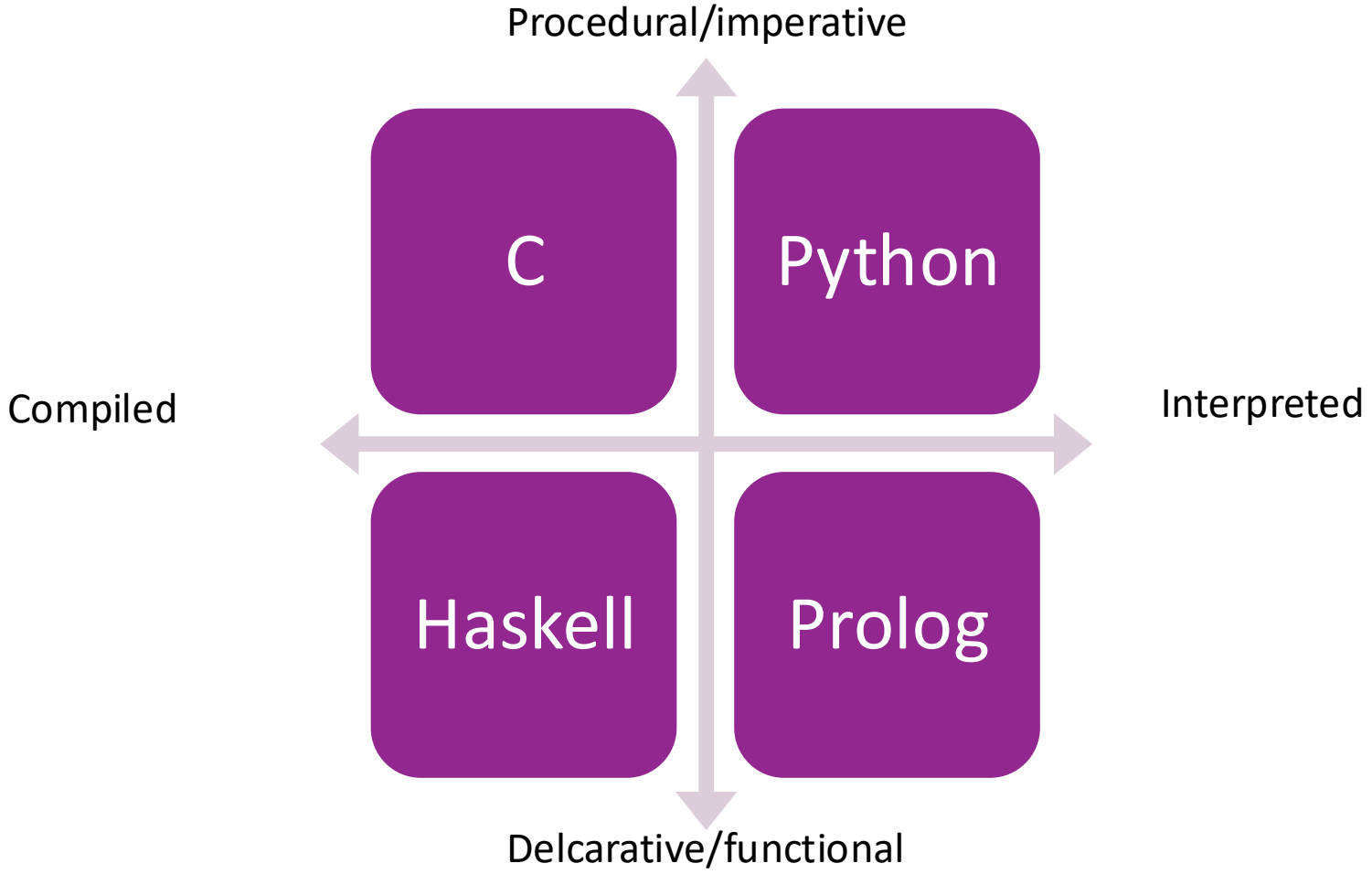
How is code in a programming language run, then?

- A language itself does not specify how the code should be run
  - The language only describes what constitutes a valid program
- Separate programs are used to convert code written into something the computer can execute

```
assignment:
    | NAME ':' expression ['=' annotated_rhs ]
    | ('(' single_target ')'
         | single_subscript_attribute_target) ':' expression ['=' annotated_rhs ]
    | (star_targets '=' )+ (yield_expr | star_expressions) !'=' [TYPE_COMMENT]
    | single_target augassign ~ (yield_expr | star_expressions)
```

https://docs.python.org/3/reference/grammar.html

Types of programming languages



Compiled languages

- Whole program converted to lower-level language
- Executed only after that static binary is generated
- Some degree of variance
  - C/C++ compile all the way down to binary from the start
  - Java/C# compile first to an intermediate bytecode, then the bytecode is compiled to binary as the program runs
- Because compilation is done early and done once, programs in these languages are often faster

Interpreted languages

- Each statement is translated down to binary in real time as the program runs
  - Python does this
  - The interpreter determines what the binary should be for every new line
- Because the translation to binary is always occurring, these kinds of programs often run slower
  - But, they are often easier to write!

Imperative/procedural

- Give the computer a series of instructions on how to complete a task
- Program may have different **state** at different points in time

```
def calculate_hypotenuse(a, b):
    c = (a**2 + b **2)**(1/2)
    return c
```

```
>>> calculate_hypotenuse(3.0, 4.0)
5.0
```

Declarative/functional

- Describe to the computer what the correct answer looks like
- Computer then applies functions, logic, etc. to find answer
- State is often not stored

```
hypotenuse(A, B, C) :- C is (A**2 + B**2)**(1/2)
```

```
[3]  ?- hypotenuse(3.0, 4.0, 5.0).
true.

[3]  ?- findall(C, hypotenuse(3.0, 4.0, C), Bag).
Bag = [5.0].
```

Some opinions on programming languages

- For scientific programming, interpreted procedural languages are usually easiest to work with and learn
  - Don’t have to worry about compilers, memory allocation, etc.
  - Scientists usually write prototypes and then leave them as prototypes
- For final products, compiled languages are often better (speed and some fewer classes of bugs)
- Some languages, such as Julia, offers a nice compromise of being easy to write like interpreted languages but with the speed and safety of compiled languages
  - Fewer resources to learn than Python, and often better as your second language than your first

Program state and variables

- Variables
  - Lower-level: Represent some kind of memory or storage location
  - Higher-level: Represent some kind of labeled information
- State
  - The content of all variables at a given point
- Variables have types (e.g., integer, string) and values (e.g., 10, “ten”)
- Variables may or may not require values to be of certain types
  - Python does not
  - Java does

Variable typing

VALID PYTHON CODE

```
a = 10
a = 'ten'
```

INVALID JAVA CODE

```
int a = 10;
a = "ten";
```

Static, strong typing

- Can reduce bugs
- Type is determined and checked during compilation
- Only certain kinds of retyping allowed

Dynamic typing

- Type of variables determined at runtime
- Type inconsistencies can only be caught during execution, which may be two years later…
- Sometimes, languages will silently "resolve" type incompatibilities
  - JavaScript: `10 + 'ten'` yields `'10ten'` rather than throwing an error
  - Python: `10 + 'ten'` gives an error

Dynamic typing and data science

- Generally, dynamically typed languages are more prevalent in data science
  - Lets you see data as generic
  - Often, you won't want to know exactly what type of data you will get
- Dynamic typing more easily allows for more generic code

What to know about Python

- Interpreted, dynamically typed
- Relies on modules and packages
  - Python beginners often suffer from not being able to import packages correctly
- Two different main versions
  - 2.* (obsolete but obstinately still around)
  - 3.* (only use this one for new code)
  - Sometimes need to specify which version you want to run

Programming activity: Printing

Basic instructions

1. Open VSCode and create a new Python with .py ending (call it whatever you like)
2. In the .py file, change the program to print the following:
   a) Your name
   b) A sentence
   c) A 4x4 square where the edges are made of the character "*" (You may need to call the `print` function multiple times)
   d) As a challenge, a picture made up of text characters (e.g., an "A" made up of "#")