Week 2

AAM

Dhruva V. Raman

Packages augment languages

```
1  import numpy as np
2  np.array([1,2,3])
```

```
array([1, 2, 3])
```

`array` is *exported* by numpy

numpy's alias is np

Principle:

- Only load functions you need
- Avoid confusion from too many function names

Never do this

```
1  import numpy as *
2  array([1,2,3])
```

Every function in numpy exported without namespacing

What happens if you have a function with the same name?

```
1  def array(v)
2    # does stuff
3    return ...
```

```
1  import numpy as np
2  np.array([1,2,3])
```

Julia has a different culture

Every function in numpy exported without namespacing

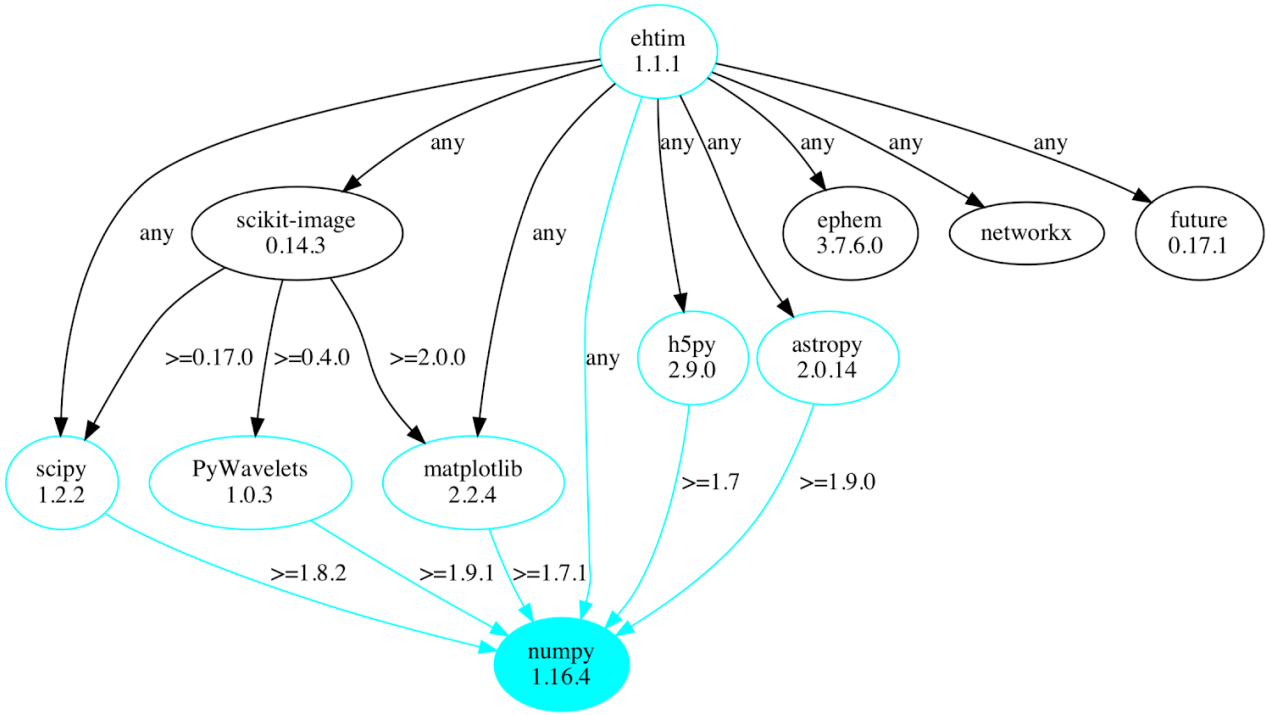
```
1 using LinearAlgebra
2 # equivalent to import in Python
3
4 eigen == LinearAlgebra.eigen
```

```
True
```

What happens if you have a function with the same name?

```
1 eigen
2 #for my func
3 LinearAlgebra.eigen
4 #for package func
```

Packages depend on packages

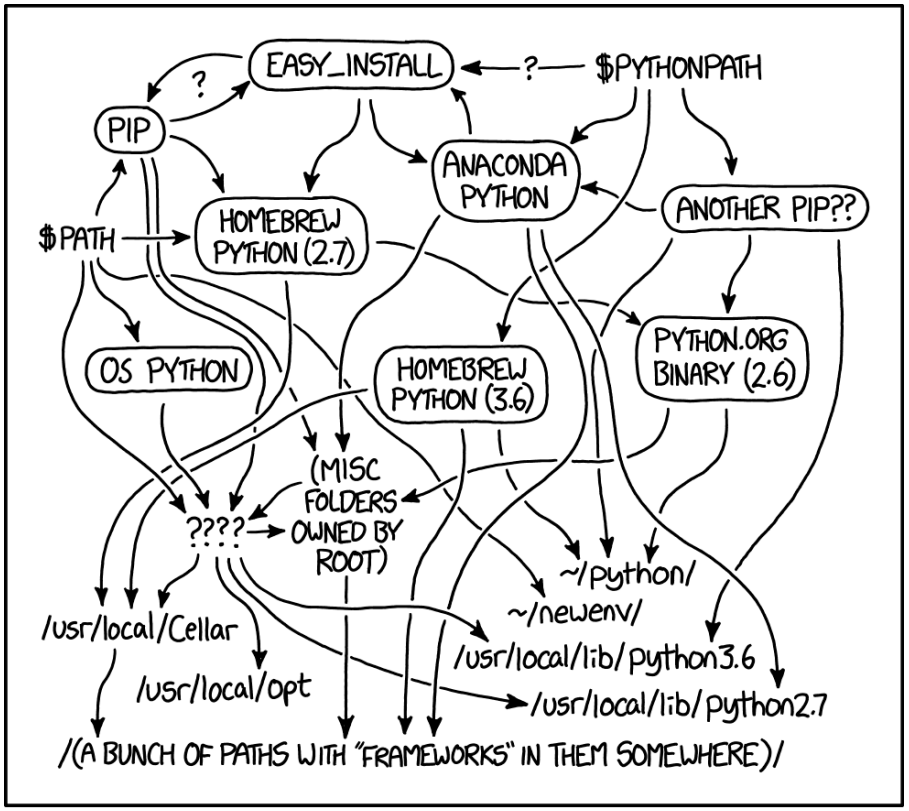


Broken by package updates

Broken by Python updates!

How come anything works?!

Package management



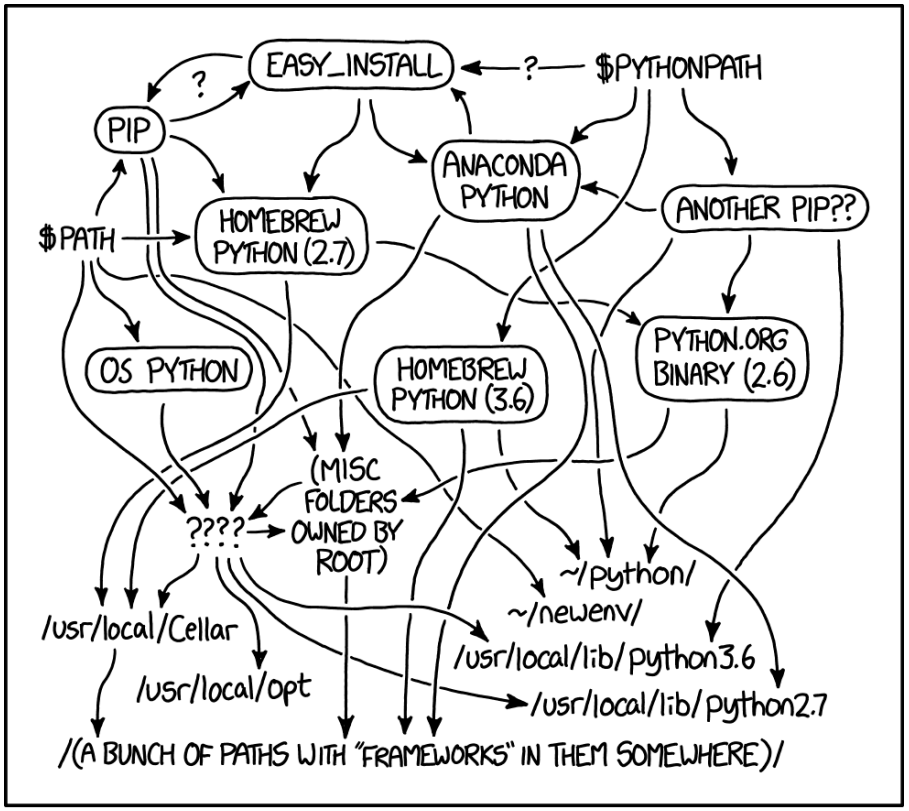
MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

Environments are walled gardens holding few, compatible versions

```
1  uv init MyProj
2  uv add Marimo
3  uv add Numpy
```

`uv` (or `pip`, `conda`, …) calculates correct dependencies to load

Package management



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

Isolate your Python/Julia projects in different environments.

Python module deals with how to. (I recommend uv manager)

Laziness now causes serious pain later

Zero-indexed languages

… are annoying

```
1  ## Python
2  A = np.array([1,2,3])
3  A[2]
```

```
np.int64(3)
```

- Python is zero-indexed
- Julia is one-indexed

```
1    ### Julia
2    A = [1,2,3]
3    A[2] #?
```

```
2
```

Common source of bugs!

Zero-indexed languages

… are annoying

Zero-indexed	One-indexed
Python	Julia
PHP	Fortran
Java	Matlab
C / C++	
Ruby	

Why?!

Zero-index:

offset from first element

One-index:

Counting number of elements

Goals today

1. How do computers do maths

2. How do we make computer maths efficient

Why?

Getting the answer is not enough

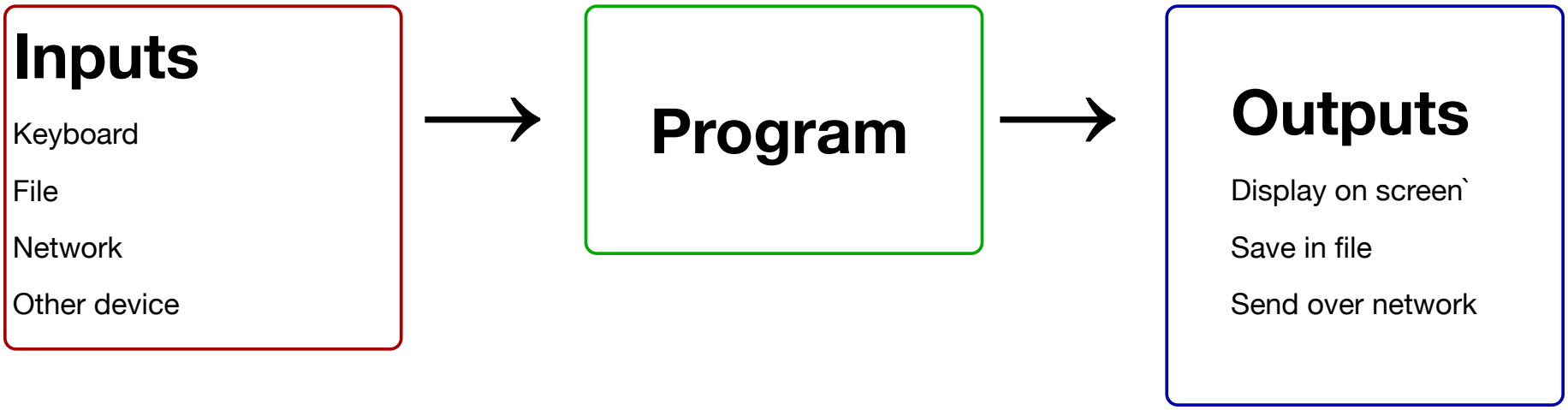
Writing an efficient algorithm is common bottleneck

Start quest for improvement soo rather than later

Structure

1. What are programs? How are they run?
2. Interpreted vs compiled languages
3. Memory models for programming
4. Allocations
5. Types

What is a program



CPU (Central processing unit)

What is a program?

Program

Inputs $\rightarrow$

Basic maths operations

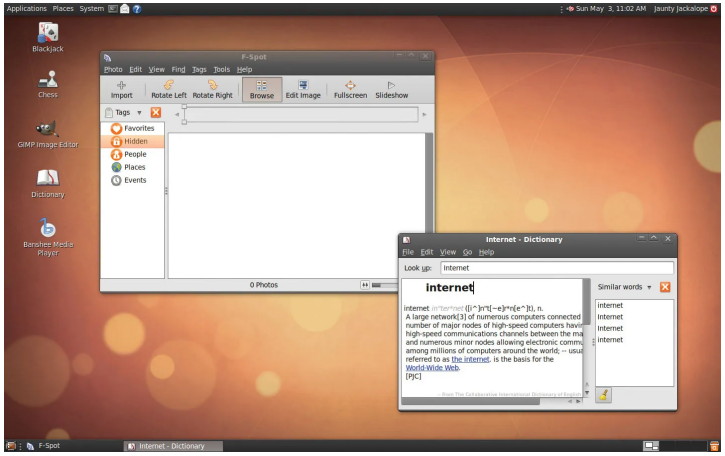
$(+, -, \times, \ldots)$

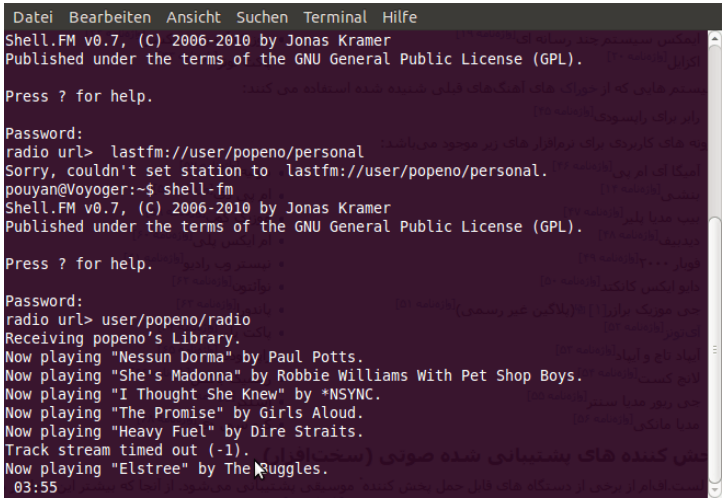
$\rightarrow$ Outputs

Conditional execution (run … if … is true)

Repetition Run things repeatedly, with some variables changing

How do we run a program?





GUI (Graphical user interface)

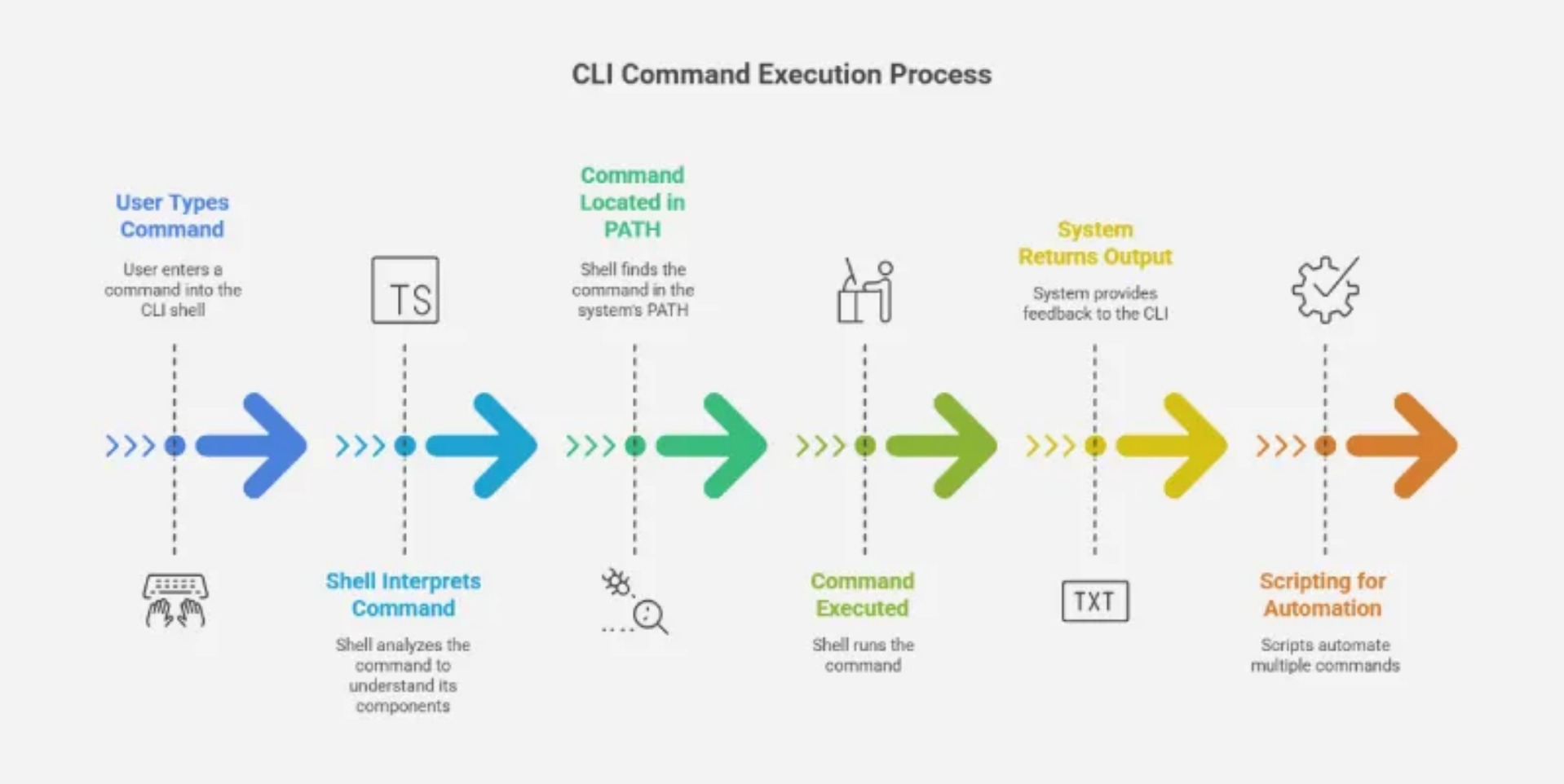
- Point and click!

CLI (Command line interface)

(/shell /terminal)

- Type program name
- Annotate with program input

How do we run a program?



How do we run a program?

```
python myscript.py
```

```
man marimo export
marimo export ipynb notebook1.py -o notebook1.ipynb
```

```
command -short_option_1 -short_option_2 --long_option
```

Learn to talk to your shell

- Spend time automating repetitive processes
- Pays off in long run!
- Unix shell is more standard (Windows subsystem for Linux)

Example: marking

List of student names and dictionary:

`names -> student numbers`

- Make folder for each student
- Populate with marking sheets, reports, initialised with correct name/number

Option 1:

Carpal Tunnel syndrome

(`ctrl-c`, `ctrl-v`)

Option 2:

Build/execute *shell script*

```
./populate_folders.sh
```

Option 3:

Run python program in shell

```
python populate_folders.py
```

What is a shell script?

```
 1 #!/bin/sh
 2 # This is a trivial shell script.
 3 echo Hello, I am a test shell script.
 4 echo
 5
 6 # Who and where are you?
 7 echo -n "Your userid is "
 8 whoami
 9 echo and you are logged into $HOSTNAME.
10 echo
11
12 # Is anyone else using your machine?
13 echo Who else is using your computer?
14 w
15 echo
16
17 # Date and to-do list
18 echo -n "Today is "
```

List of instructions that CLI performs sequentially

Can move and transform data across multiple programs!

What is a script?

```
1  x=4
2  y=5
3  z=x+y
```

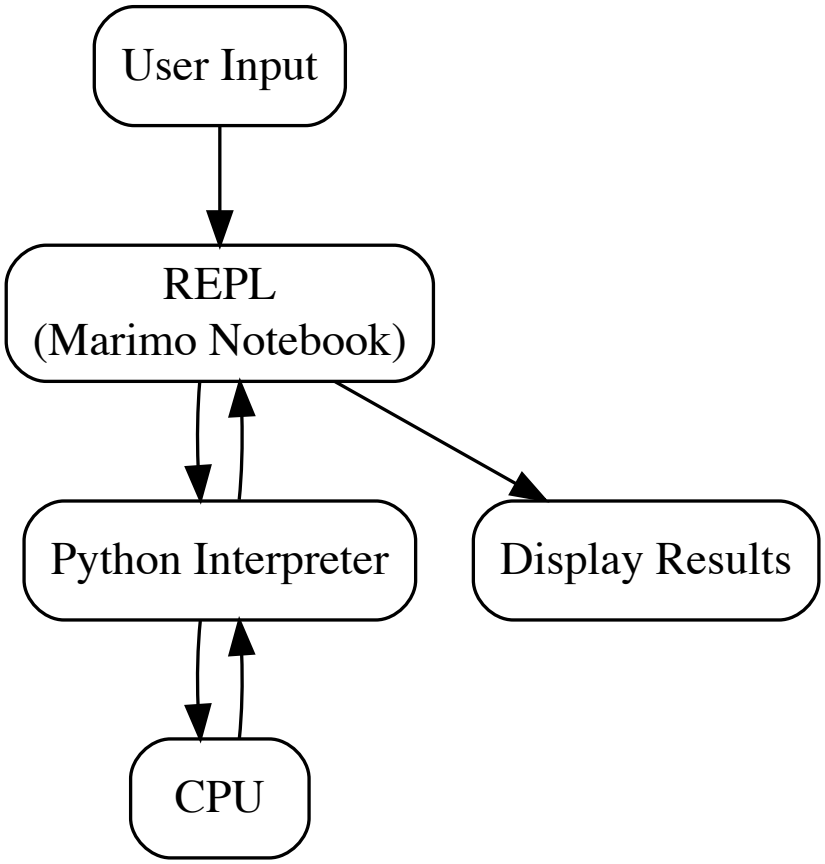
9

```
1  x=4
2  y=5
3  z=x+y;
```

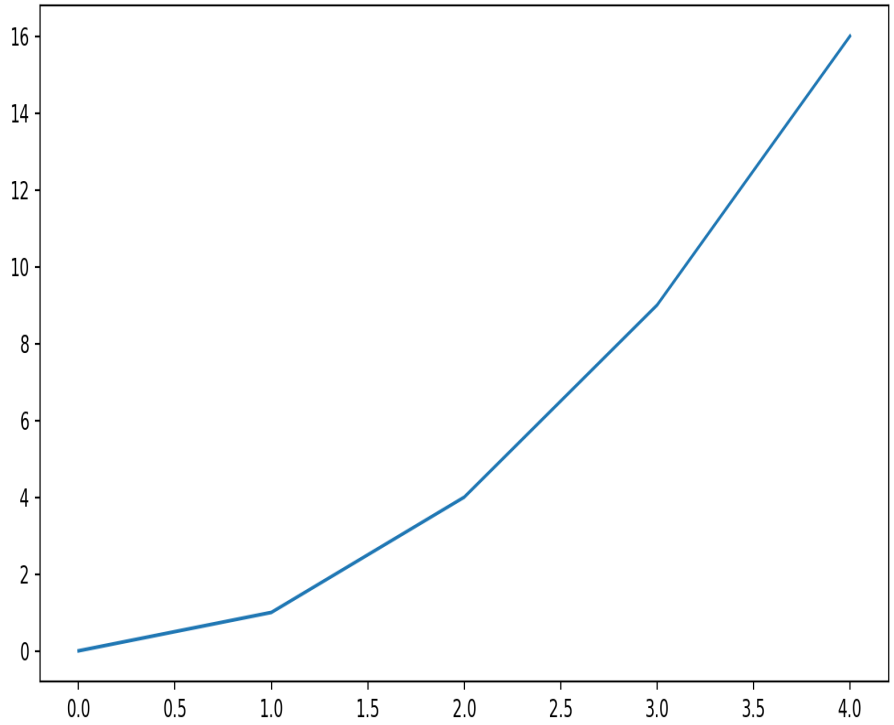
List of instructions with no defined input or output

Suppress visible output (last executed line) with a semicolon

Scripts are interpreted by a REPL program




```
1 x = [0, 1, 2, 3, 4]
2 y = [0, 1, 4, 9, 16]
3 import matplotlib.pyplot as plt
4 plt.plot(x,y) #input
```



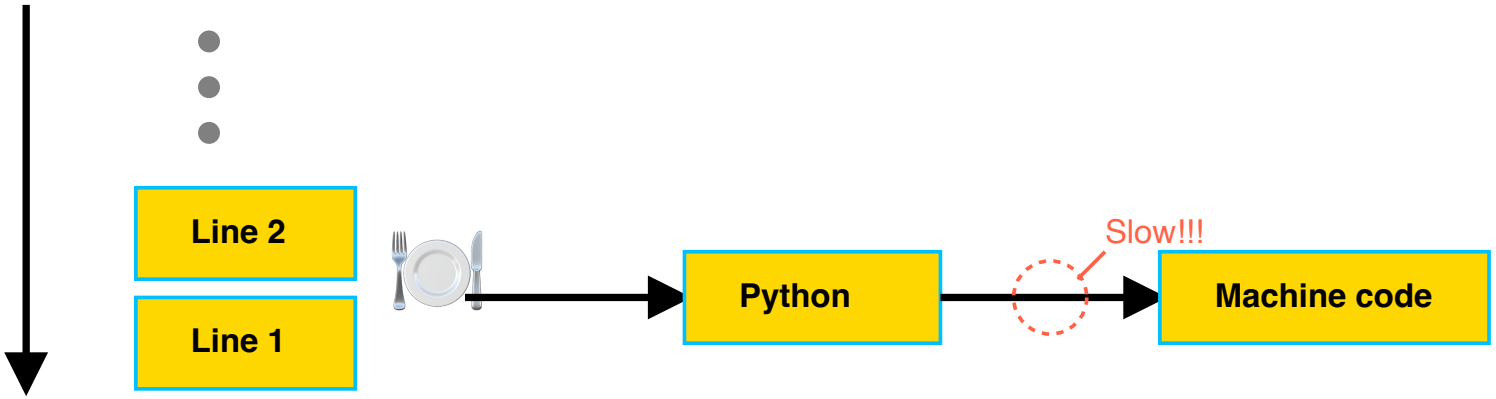
REPL scripts are for exploratory programming

Language	REPL
Python	IPython3
Julia	Julia REPL
eg C++	none
eg Rust	none

Notebooks are essentially a hidden, prettified REPL

In other modules you might interact directly with IPython3

Interpreted languages

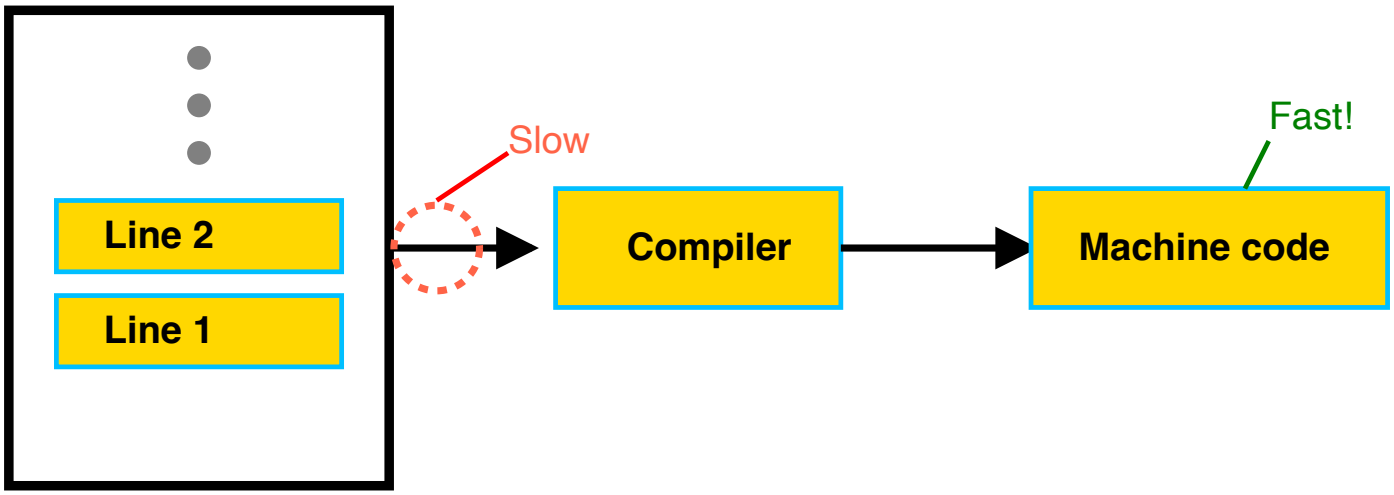



```
1  for i in range(1000000):
2    computation(i)
```

Interpreting each line takes time.
$\times 100000$

Compiled languages

…are much faster




```
1  function keep_adding(a::Int64)
2    for i = 1:100000
3      a +=1
4    end
5  end  #julia code
```

Julia (or any compiled language) will munch this!

Compiled language example

C / C++

```
1 gcc my_code.c -o output_file.exe
```

`gcc` is a compiler program that produces an executable

```
1 output.exe
2 #machine code!
```

`output.exe` is unreadable to humans. Specific computer will run it.

Production software is written in compiled languages

…but this isn’t a software development course

Always use compiled languages?

```
1 gcc my_code.c -o output_file.exe
```

Every change needs recompilation!

Recompilation induces delayed feedback

Compiled languages are harder to learn

Buying computing power is cheaper than training a programer

Why does … use Python?

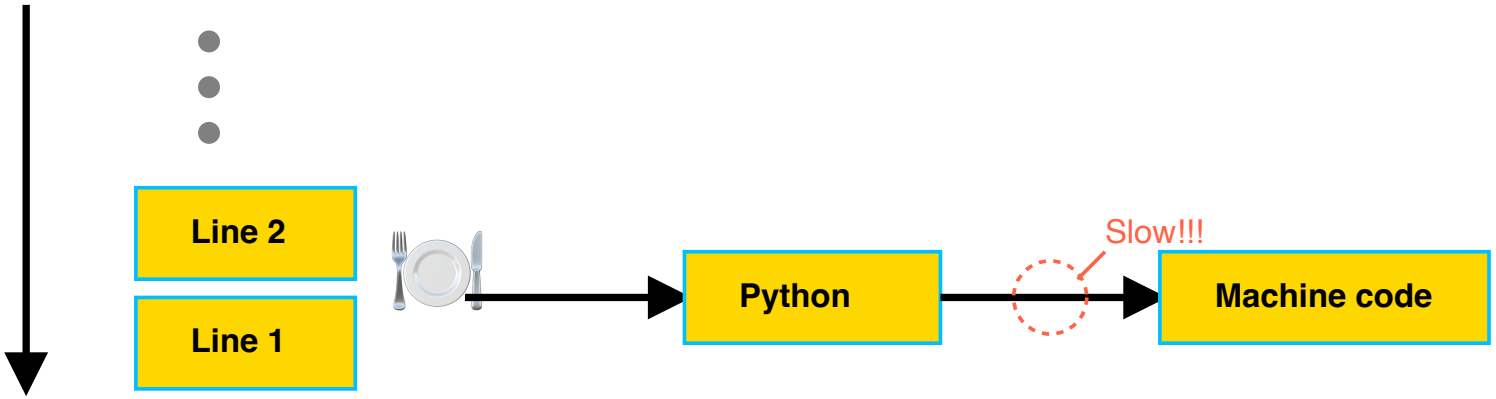
$$\begin{bmatrix} 1 & 3 & 4 & \dots \\ 2 & 7 & 6 & \dots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \times \begin{bmatrix} 5 & 7 & 2 & \dots \\ 2 & 5 & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

```
1  numpy.matmul(m1, m2)
```

`matmul` is a program (written in C++, a fast language)

…needs lots of lines of code?!

Why does … use Python?



Ideally each line is a specialised program doing lots of work

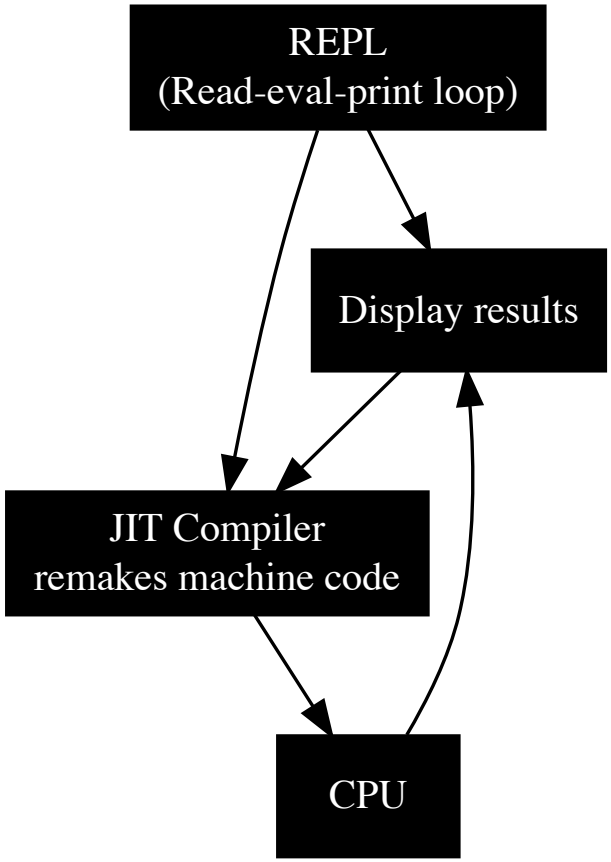
Python is a glue language

Just-in-time compilation

Quacks like an interpreted language

Walks like a compiled language

```
1 import jax.numpy as jnp
2 def square(x):
3     return x * x
4
5 square_jit = jax.jit(square)
6
7 x = 3.0
8 print(square(x))
9 print(square_jit(x))
```



Just-in-time compilation

JIT	Interpreted
Write algorithm yourself	Glue algorithms from specialised programs
Give compiler lots of info	Minimise lines of code

```
1 import jax.numpy as jnp
2 def func(x):
3   for i in range(100000)
4     x +=1
5
6 func_jit = jax.jit(func)
```

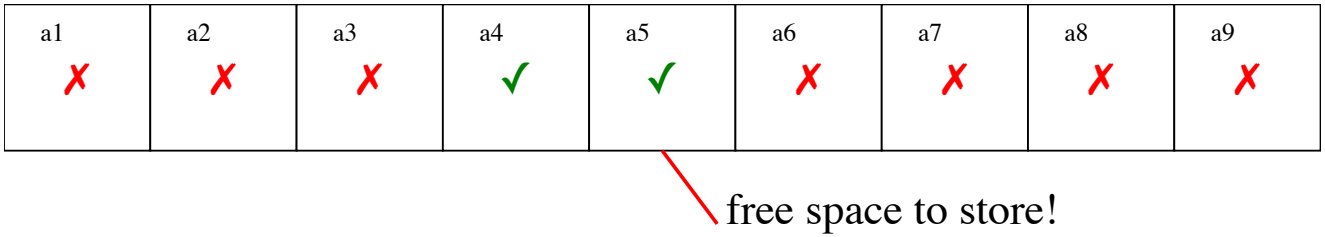
Most ML frameworks use JIT modules

```
1 import pytorch as torch
2 import tensorflow as tf
3 import jax.numpy as jnp
```

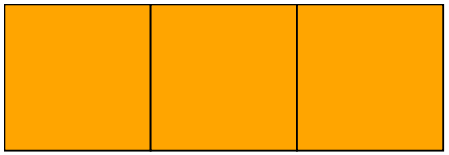
Increasingly useful to understand principles of JIT programming

Memory models in programming

```
1  var = 4
```



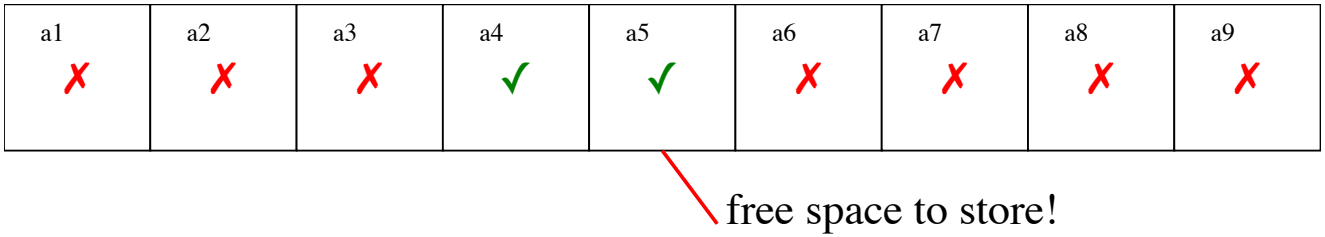
Each memory location has an address

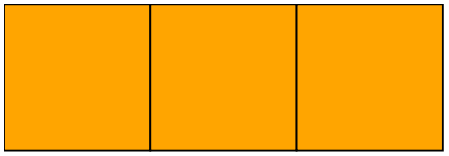


variable

`var` won't fit!

Lots of variables?





variable

Where do I put them?

Takes time to find space

Compiler does organisation during compilation

Preallocation is good

```
1  result = []
2  for i in range(1000000):
3      result.append(i * 2)
```

Each iteration, find a new (larger) container! ✗

```
1  size = 1000000
2  result = np.empty(size, dtype=int)
3
4  for i in range(size):
5      result[i] = i * 2
```

Single allocation!

For loop mutates

Size of each element fixed (`int`)

Pass by reference

```
1 arr1 = np.array([1, 2])
2 arr2 = arr1
3 arr2[0] = 10
4 #arr1?
```

```
array([10,  2])
```

Both variables access same block of memory

Think of `arr1`, `arr2` as shortcuts to the same website

Changing website content changes output of both links

Pass by value

```
1 import copy
2 # standard library don't install
3
4 arr1 = np.array([1, 2])
5 arr2 = copy.deepcopy(arr1)
6 arr2[0] = 10
```

`arr2` is a physical copy of contents of `arr1`

Think of `arr2` as printing a new website

Further changes to `arr2` won't affect `arr1`: they diverge

Pass by what?

```
1  c = 5
2  d = c
3  c = c + 1
```

Pass by what?

```
1 c = 5
2 d = c
3 c = c + 1
4 print(d)  # Output: 5
5 print(c)  # Output: 6
```

Integers are immutable

Assignment (d=c) assigns d to a new memory address

(Im)Mutable objects

Immutable Objects	Mutable Objects
`int`	`list`
`float`	`dict`
`str`	`set`
`tuple`	`bytearray`
`frozenset`	`numpy.ndarray`
`bool`	

Type annotation

```
1 def multiply(a: int, b: int) -> int
2     return a * b
```

Every variable in every language has a type

Type annotation speeds code in (JIT) compiled languages.

- Avoids real-time type conversion
- Helps compiler pre-allocate space

Only helpful for debugging in Python :(