Python for Data Science

IISER-Kolkata ML4HEP Pre-school Lecture Series
May 12, 2025 - Lecture 1

Subir Sarkar, SINP
subir.sarkar@cern.ch

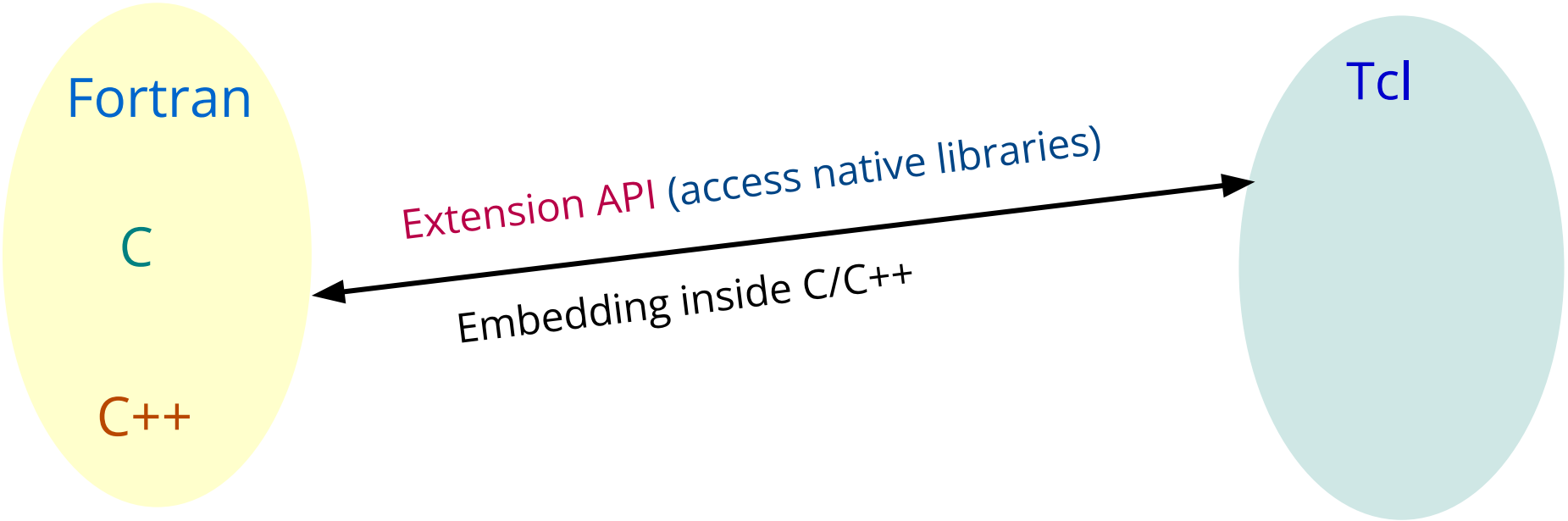
Python's place in the computing world

Major Programming Languages in the Scientific World

Compiled

Byte Compiled
(partly compiled + interpreted)

Interpreted



- static type
- fast
- source level portability

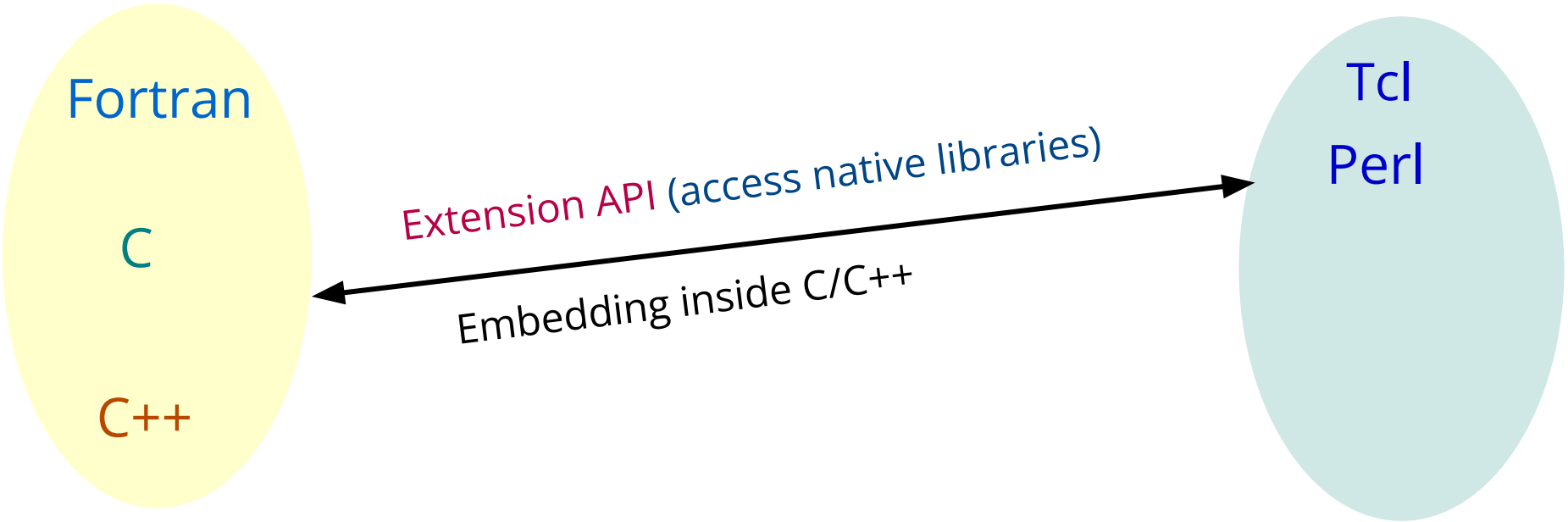
- dynamic type
- portable
- relatively slow
- scripting/embedding
- interface to C/C++/Java

Major Programming Languages in the Scientific World

Compiled

Byte Compiled
(partly compiled + interpreted)

Interpreted



- static type
- fast
- source level portability

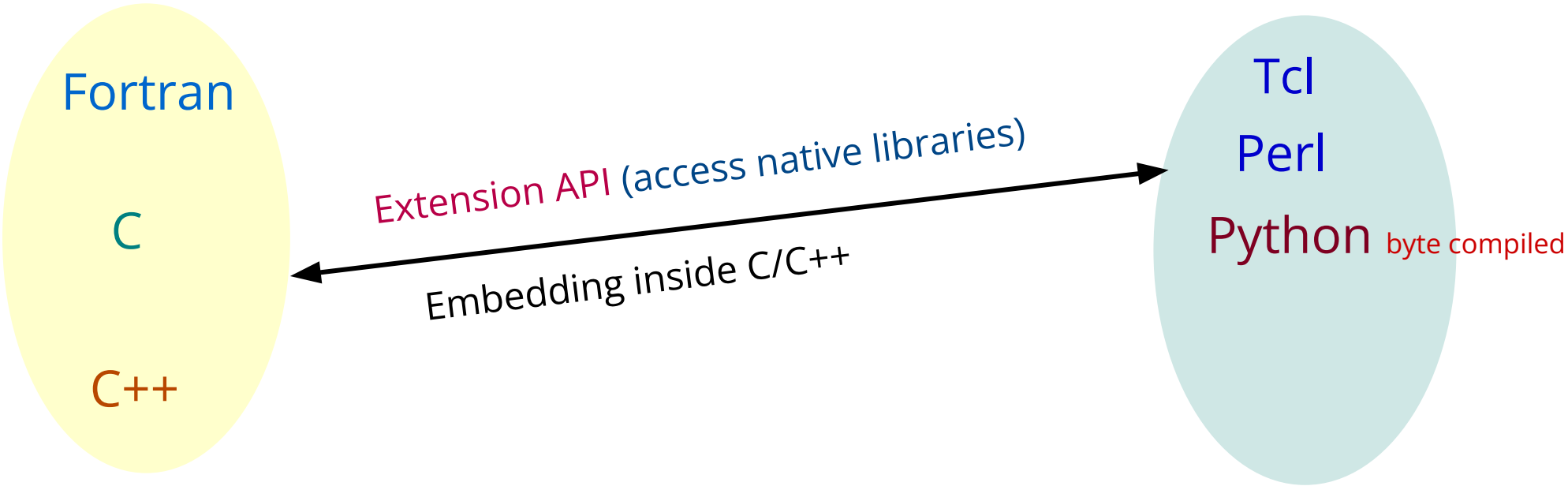
- dynamic type
- portable
- relatively slow
- scripting/embedding
- interface to C/C++/Java

Major Programming Languages in the Scientific World

Compiled

Byte Compiled
(partly compiled + interpreted)

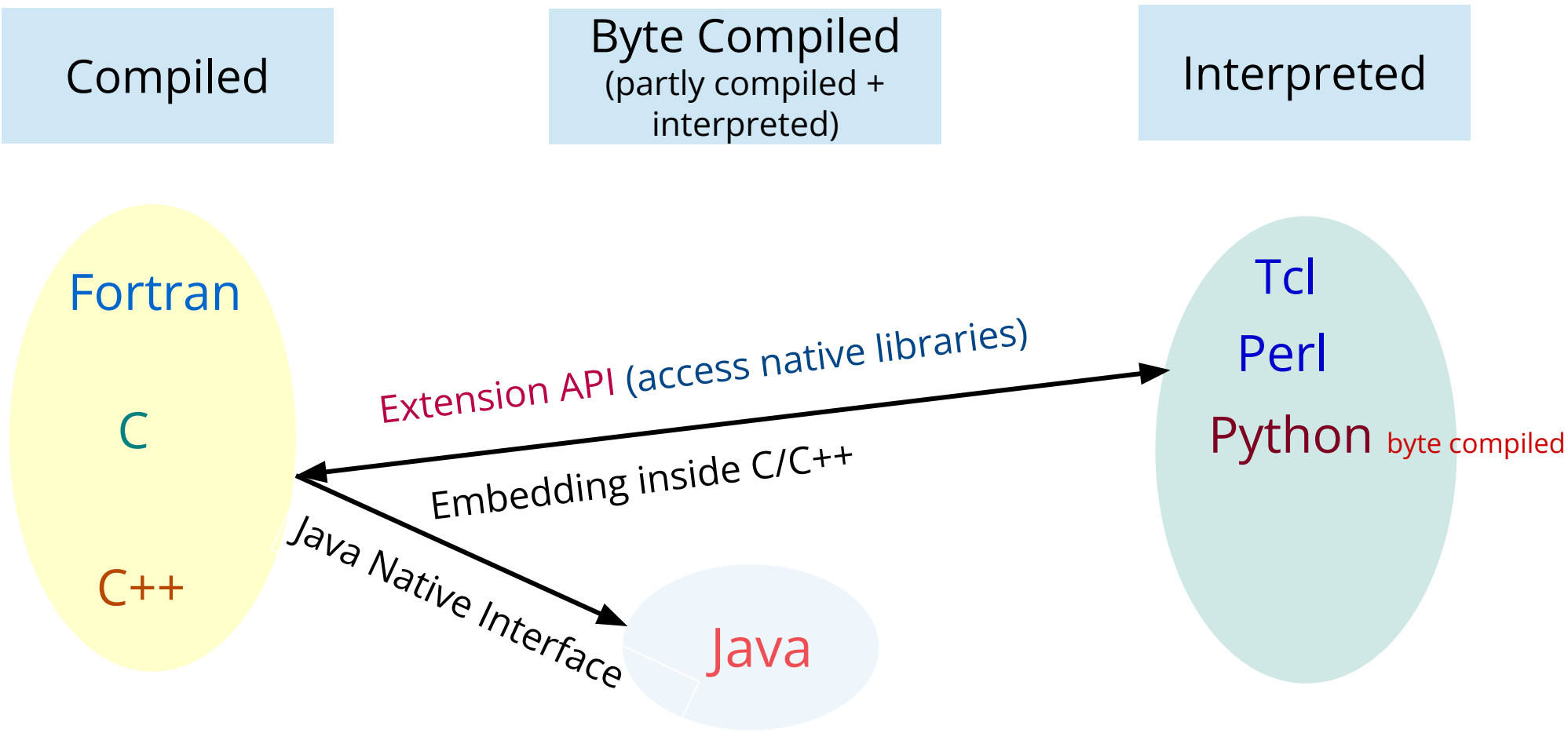
Interpreted



- static type
- fast
- source level portability

- dynamic type
- portable
- relatively slow
- scripting/embedding
- interface to C/C++/Java

Major Programming Languages in the Scientific World

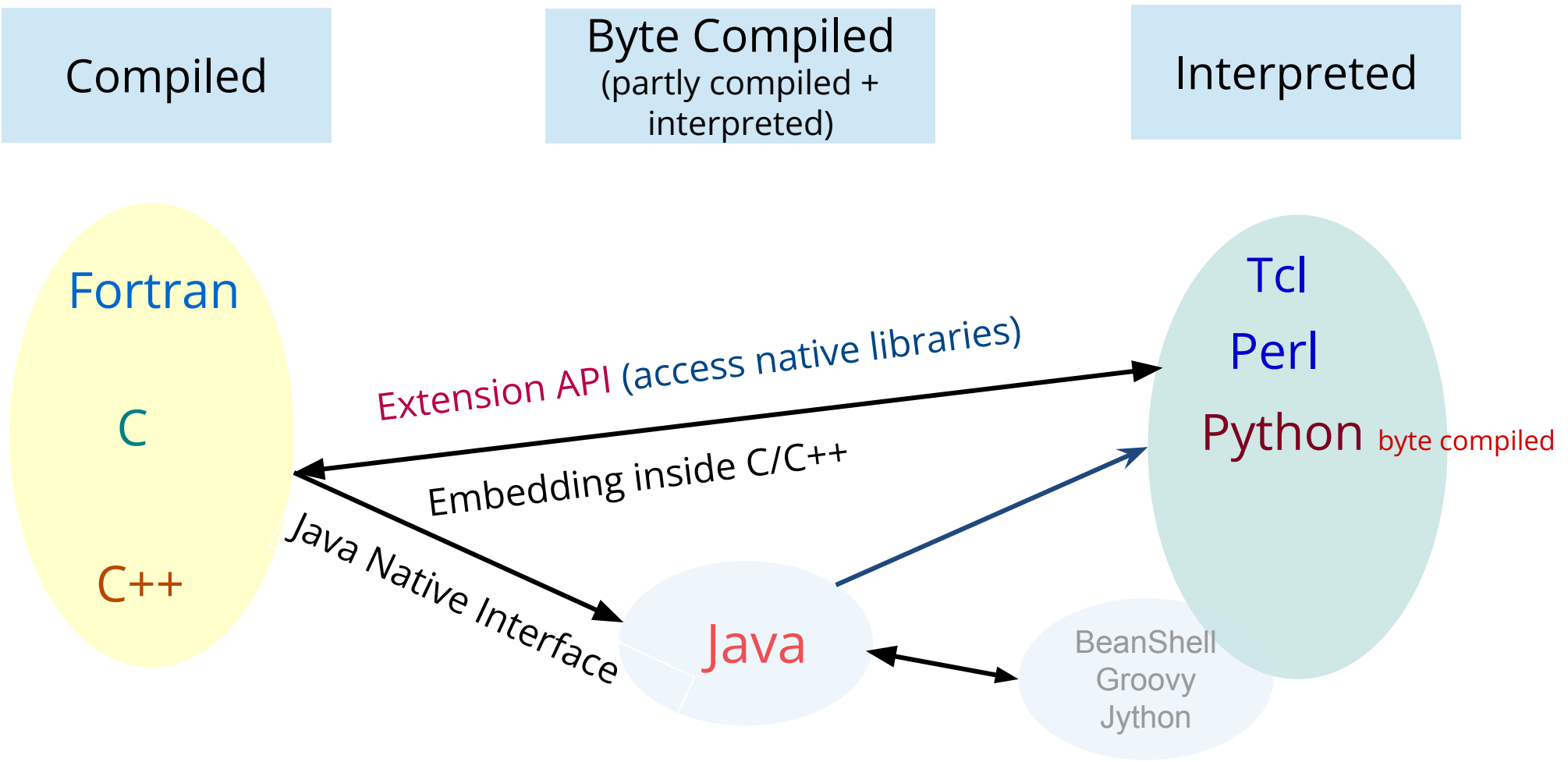


- static type
- fast
- source level portability

- static type
- secure
- portable

- dynamic type
- portable
- relatively slow
- scripting/embedding
- interface to C/C++/Java

Major Programming Languages in the Scientific World

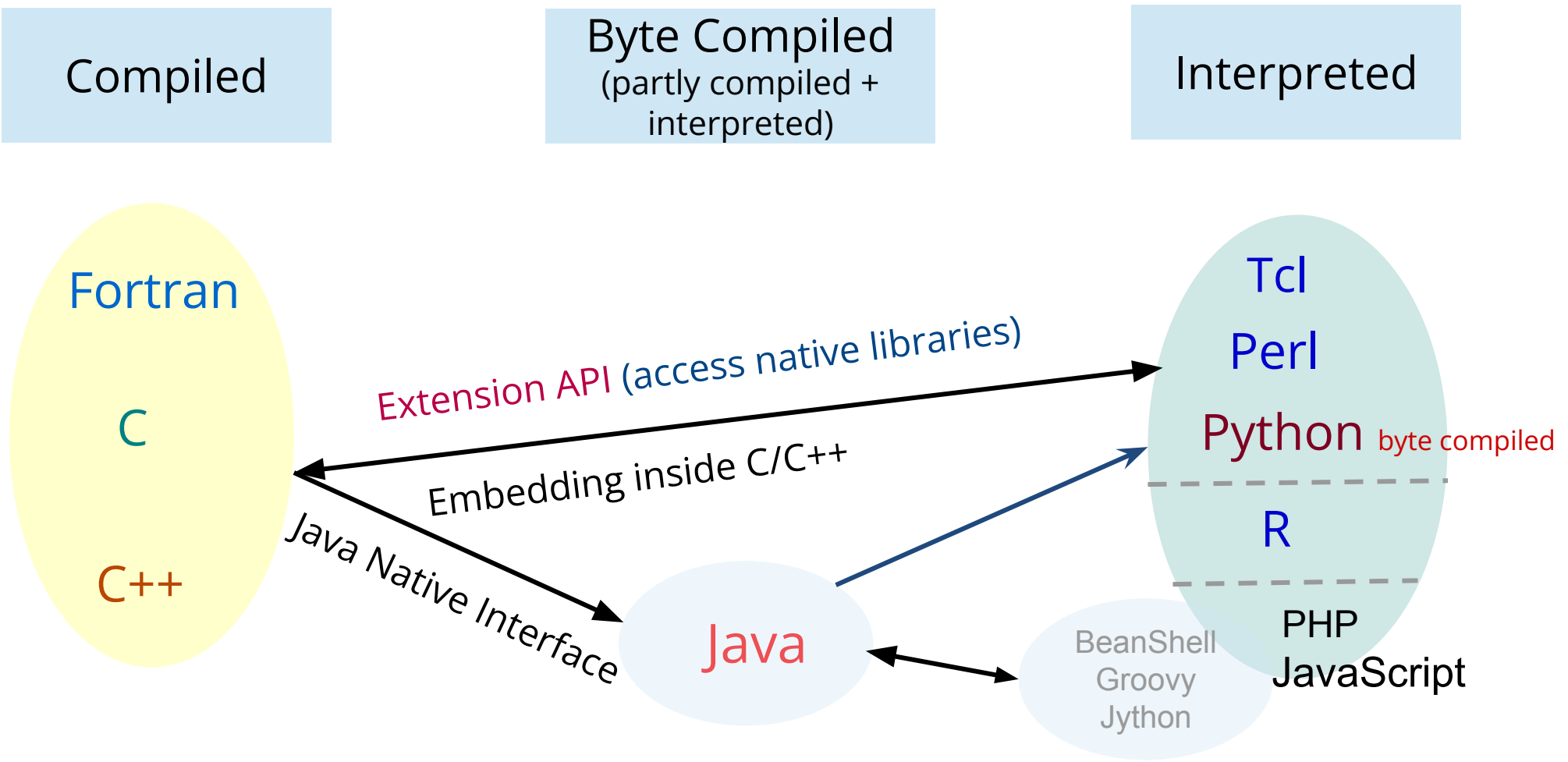


- static type
- fast
- source level portability

- static type
- secure
- portable

- dynamic type
- portable
- relatively slow
- scripting/embedding
- interface to C/C++/Java

Major Programming Languages in the Scientific World

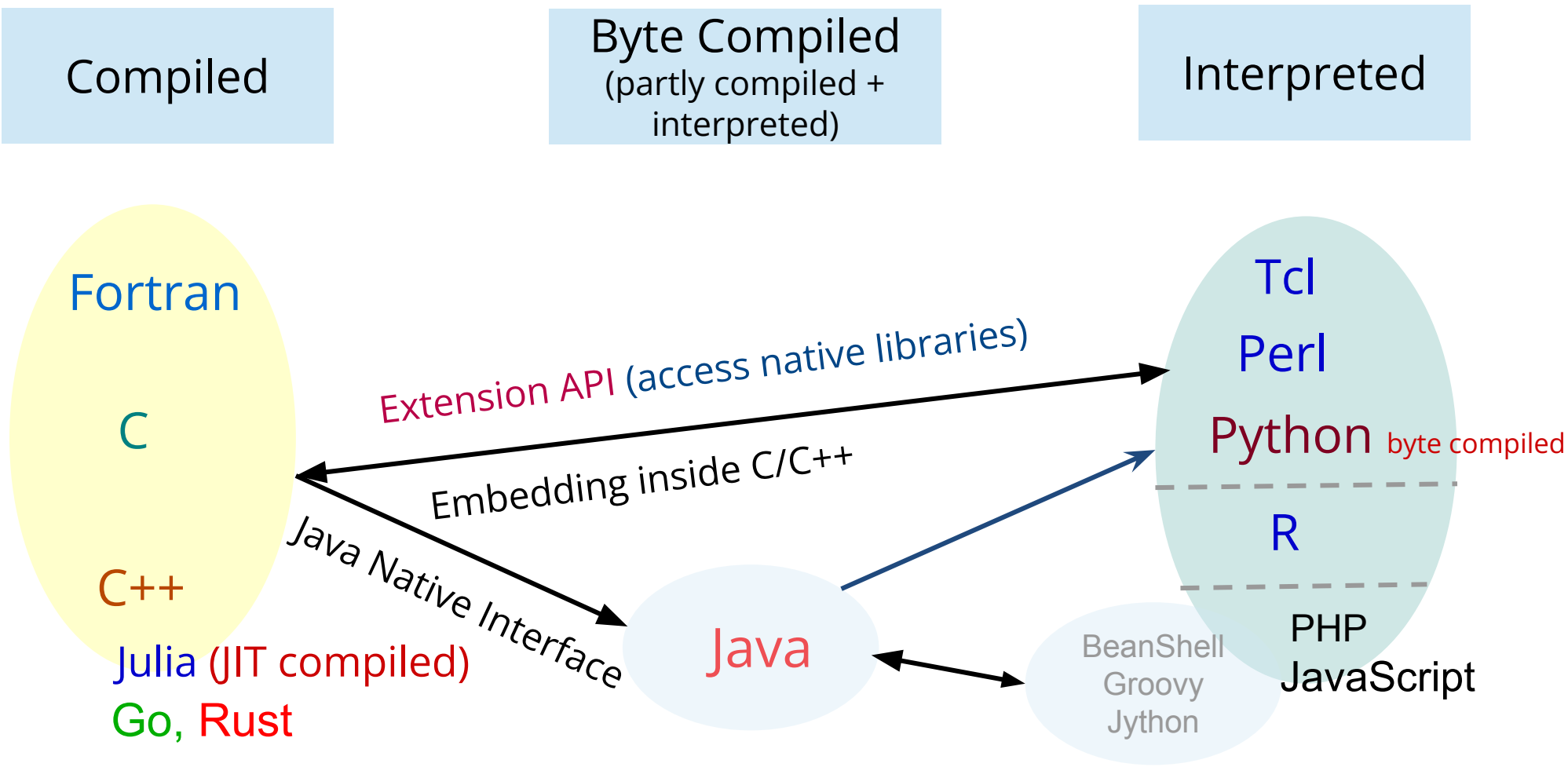


- static type
- fast
- source level portability

- static type
- secure
- portable

- dynamic type
- portable
- relatively slow
- scripting/embedding
- interface to C/C++/Java

Major Programming Languages in the Scientific World



- static type
- fast
- source level portability

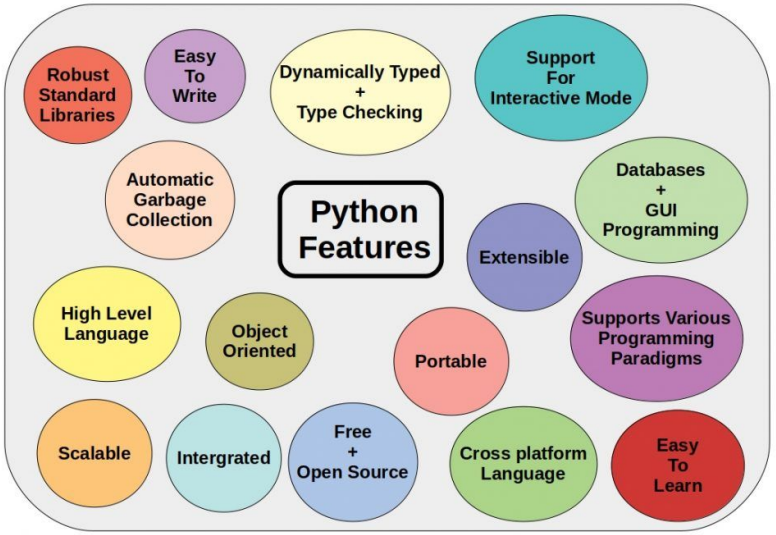
- static type
- secure
- portable

- dynamic type
- portable
- relatively slow
- scripting/embedding
- interface to C/C++/Java

Python at a glance

Python Design Highlights

- Supports all major programming paradigms
  - procedural, object oriented and functional
  - imperative and declarative
- Simple syntax, white space important, pass by-reference
- Dynamically typed, loosely bound
- Scripting & driver language
  - binds different components together
  - seamlessly supports multi-language environment



https://starship-knowledge.com/awesome-python-data-science-libraries

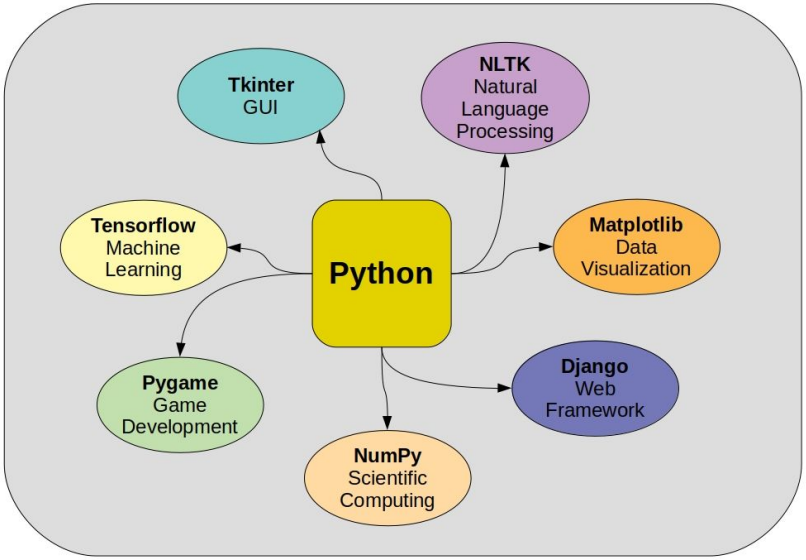
- One can programmatically build and execute Python code as string on-the-fly, inside a running Python program
- Object Orientation from ground up (no data hiding, though!)
- A very high level language (VHLL)
  - built-in support for exception
  - introspection is an integral part of the language
- High level API to seamlessly access C/C++/Java/Fortran libraries
  - opens up infinite possibilities
  - Python objects can be directly created from the native library (.so)

Integrated Development Environment

- Interactive environment
  - python shell
  - ipython (https://ipython.org)
  - jupyter notebook - https://jupyter.org/
  - IDLE, spyder, PyCharm
  - SWAN - https://swan.web.cern.ch/ - a platform to perform interactive data analysis on the cloud

Python application domain

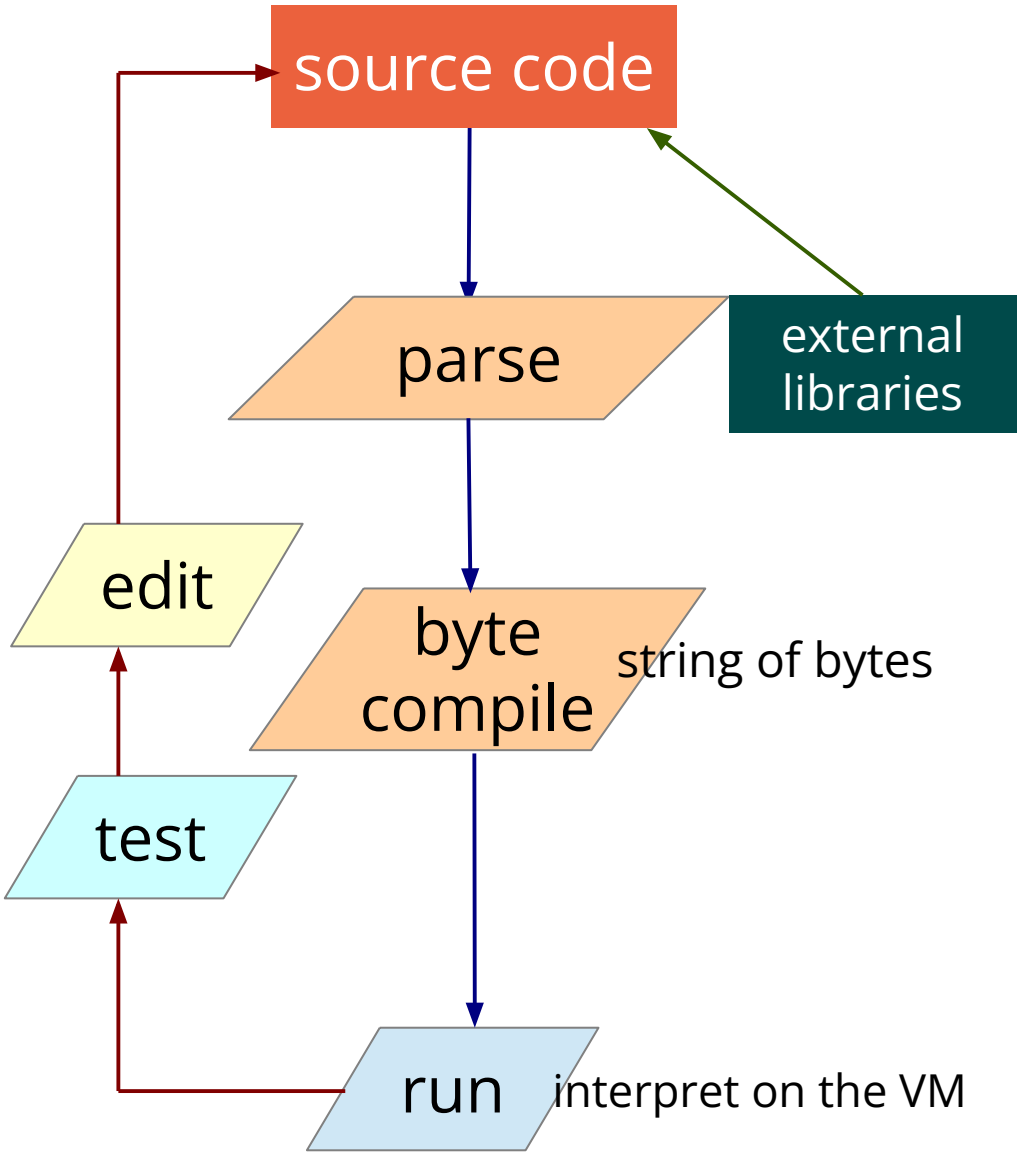
- Scientific computing, data science
  - NumPy, pandas, PyTables, PyROOT, Machine Learning frameworks (classical, quantum)
- System administration
- Web frameworks
  - CGI, CherryPy, Flask, Django, TurboGears and a hell lot more
- Interface to databases, XML and json parsers
- GUI development
  - PyGtk, PyQT/PySide, wxPython
- Game development



https://starship-knowledge.com/awesome-python-data-science-libraries

Python as an interpreted language

- A Python interpreter
  - parses a Python program/source file (.py)
    - compiles into byte-code (.pyc)
  - executes byte-code
    - interprets on a virtual machine (VM)
  - no need for compilation and linking phases
    - no binary executable



https://www.datacamp.com/community/tutorials/data-science-python-ide

Portability & Performance

- Python source code is inherently portable across platforms/architectures
  - the underlying virtual machine (VM) instance takes care of architecture dependence
  - in most cases you pay a performance penalty
    - vis-a-vis Fortran/C/C++/Go/Rust(/Julia)
    - benefits usually outweigh lack of performance
    - there are alternative implementations of Python to address performance issues with certain limitations
      - Cython - uses additional C/C++ extension libraries
      - PyPy (w/ JIT) - works with pure Python, not with C extension
      - numba (w/ JIT) - sprinkle decorators to accelerate code execution

Interactive Python

Interactive Python

```
$ python
>>> credits
>>> copyright
>>> help(1) # help on integer
>>> help()
help> keywords
help> quit
>>>
```

Get information, extensive help

There are more than one ways to exit an interactive session

- `>>> CTRL-D`
- `>>> quit()`
- `>>> raise SystemExit`
- `>>> import sys`
- `>>> sys.exit()`

Python as a calculator

```
$ python3
Python 3.8.10 (default, Mar 15 2022, 12:22:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 2 + 2
4
>>> a = 2
>>> b = 2
>>> c = a * b
>>> print(c)
4
>>> bin(16)
'0b10000'
>>> pow(2,3)
8
>>> log(10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'log' is not defined
```

Python does not load the full `math` library in memory

- Two major releases in use: python2 and python3
- python3 is the default on modern OSes, breaks backward compatibility
- python3 is used in this course

Python as a calculator

```
>>> import math  # load a module
>>> dir(math)    # see what it offers
[... 'log', 'log10', 'log1p', 'modf', 'pi',
'pow', 'radians', 'sin', 'sinh', 'sqrt',
'tan', 'tanh', 'trunc']

>>> math.pi
3.141592653589793

>>> from math import sqrt,pow
>>> pow(2,3)
8.0

>>> sqrt(10)
3.162277601683795

>>> _ # result of the last expression
3.162277601683795
```

Code Execution: script, eval, exec

Python script (on Linux)

```
#!/usr/bin/env python3

# first.py
for i in range(10):
  print(i, end=" ")
```

```
[1]$ python first.py
0 1 2 3 4 5 6 7 8 9

[2]$ chmod a+x first.py

[3]$ ./first.py
0 1 2 3 4 5 6 7 8 9
```

Find the default Python interpreter

Turn on execution bit

eval

```
$ python3
>>> result = eval('1.0 + 1.0')
>>> print(result, type(result))
2.0 <class 'float'>
>>> print(eval("len('hello world!')"))
12
>>> print(eval("__import__('os').getcwd()"))
/home/sarkar/python/examples
```

exec

- one can construct Python code as string and execute on-the-fly from within a running Python program

```
>>> code = 'print("hello world!")'
>>> exec(code)
hello world!
```

Data type, variable scope, namespace

Built-in Data Types

Data type	Example
Numbers	3.1415, 1234, 999999999, 3+4j
String	“Hello”, “guido v”
List	[1, [2,'three'], 4]
Dictionary	{'compiled': 'Fortran,C,C++', 'interpreted': 'Perl, Python, Ruby'}
Tuple	1,'world!',3,4 or (1,'world!', 2, 3)
File	f = open('myfile.dat', 'r').readlines()

Python - dynamically typed, loosely bound

- Name (refer) a variable, no need to declare the type
  - variable type is context sensitive
  - from Python 3.6 static type is allowed, basically as a hint
- Python keeps track of type of an object referenced by a variable name during it's lifetime i.e Python is NOT weakly typed

```
>>> a = False
>>> print(type(a))
<class 'bool'>
```

```
>>> a = [1,2,3,4]
>>> print(type(a))
<class 'list'>
```

```
>>> a = 5
>>> print(type(a))
<class 'int'>
```

```
>>> a = 1,2,3,4
>>> print(type(a))
<class 'tuple'>
```

```
>>> a = "Hello World!"
>>> print(type(a))
<class 'str'>
```

```
>>> a = None
>>> print(type(a))
<class 'NoneType'>
```

Variable Scope & Lifetime

- Scope
  - part of a program where a variable is accessible
- Lifetime
  - duration for which a variable is accessible
- In general, variables defined
  - in the main body of a program have global scope
    - visible throughout the file, and also inside any file which imports that file
    - global scope is usually a bad idea
  - inside a function have local scope
  - in a class have local scope, but can be accessed from outside through the object (no data hiding)

Variable Scope & Lifetime

```
# global
a = 0

if a == 0:
    b = 1

# what do we expect?
print(b)
```

```
# local
def test_scope(c):
    d = 3
    print(c, d)

# call the function
d = 5
test_scope(7)

# what do we expect?
print(c, d)
```

Type System

- static vs dynamic
  - type checking at compile time vs leaving the responsibility to runtime
- strong vs weak
  - how runtime treats types
  - 1 + “1” gives error for strongly typed language (e.g Python)
- explicit vs implicit
  - how type conversion takes place

```
>>> int("Hello")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'Hello'
>>> int('5')
5
>>> float('5.2')
5.2
```

True or False

- The empty string is mapped to False, every other string is mapped to True

```
>>> s = ''
>>> if s:
...     print('hello')
...
>>> s = 'rob'
>>> if s:
...     print('hello')
...
hello
```

- For integers, 0 is mapped to False and every other value to True
- For floating point numbers, 0.0 is mapped to False and every other value to True

```
>>> a = 1
>>> if a:
...     print('hello')
...
hello
```