Intorduction to Python

Scientific Computing

Stefan Abi-Karam

Summer 2023

Table of Contents

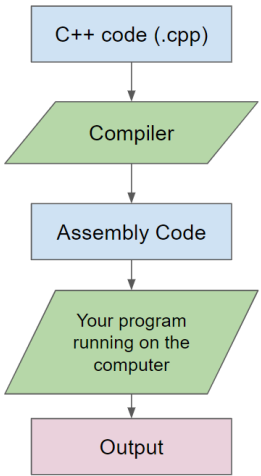
Table of Contents

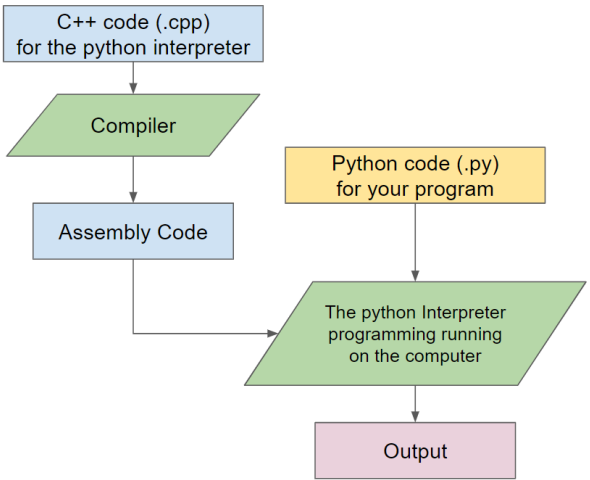
What Is Python

- Python is an interpreted, high-level, general-purpose programming language
- Interpreted means there is no python-to-assembly compiler, someone wrote and compiled a program in C (CPython) which is a program that just reads and executes your Python code
- Python also has a large standard library to achieve most tasks with relatively little code
- There are thousands of other libraries out that that other people have written that you can download and import into your code to do more complex stuff (ex. scientific computations and plotting)

Compiled Program vs. Interpreted Program

Compiled Program vs. Interpreted Program





Why Choose Python

- There is the least amount of friction from the idea in your head to executing code that works and answers your research questions
- Python has a large community across many different fields of research
  - Very strong popularity in scientific computing and data science
- This means that many people in these fields has put in effort to write and maintain python libraries specific for scientific computing and data science
- In the absence of libraries for a particular task you are attempting or area of expertise you are working in, you can easily package and publish your code as its own library.

Why Choose Python

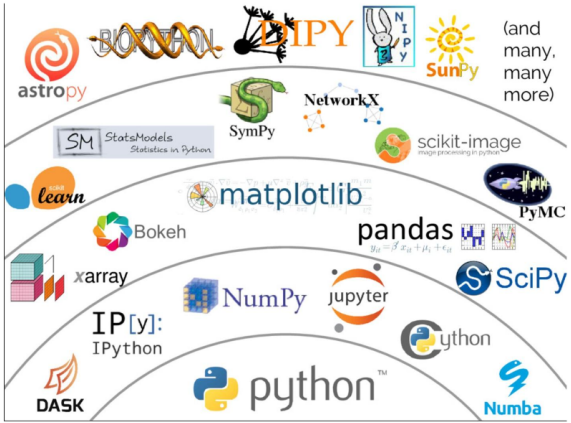


Table of Contents

Python Installation and Versions

- When you install Python you download a version of the Python interpreter
- Please use version 3.9, it has the newest features and supported by most libraries you will use
- You can install multiple versions on the same computer, you just end up with more than one interpreter you can use
- If you want to download Python you can do so at their website: `https://www.python.org/`
- There's also a program called Conda / Anaconda that used a lot in the research community to manage Python installations and other libraries you may want to install and use
- **I recommend using Conda to install Python and manage your Python environment and packages if you are going to install Python on your own computer**

Using Python

Once Python is installed on your machine you can you use the Python command to run programs in the command line

```
> python your_file.py
```

Each Python installation also comes with a package manager named `pip` which you can also run from command line

```
> pip install numpy
```

If you are using Conda, you can call the `conda` command to install packages and manage Conda environments

```
> conda install numpy
```

Table of Contents

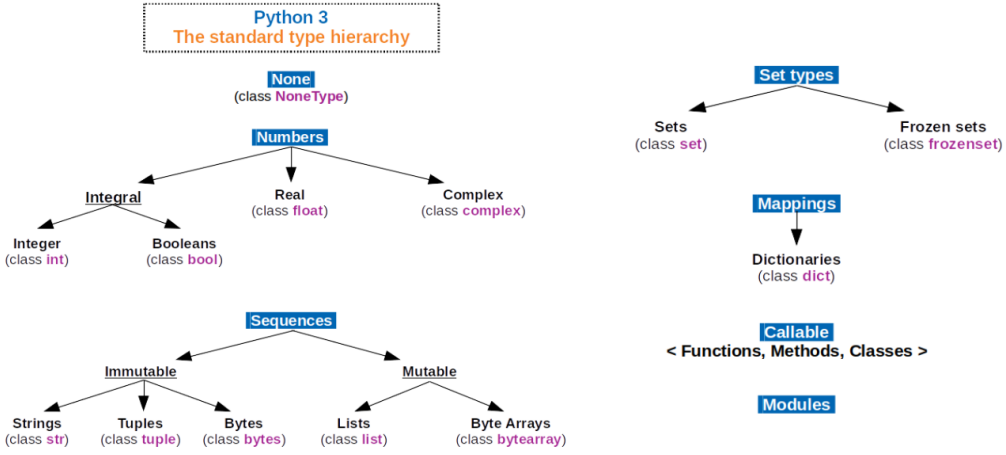
What Are Types

- Types are a way to classify objects
- Each object has a type
- Types are used to define how you can use the object

Python Types

- Numeric Types
  - `int`: Integer Number
  - `float`: Floating Point Number (decimal)
  - `complex`: Complex Number
- Sequence Types
  - `list`: List (mutable)
  - `tuple`: Like List (immutable)
  - `range`: Immutable sequence of numbers
- Text and Binary Sequence Types
  - `str`: String (text / immutable sequence of characters)
  - `bytes`: Immutable array of bytes
  - `bytes`: Immutable array of bytes
- Set Types
  - `set`: Set (same usage as mathematical set)
- Mapping Types
  - `dict`: Dictionary (new concept to most of you)
- Other Types
  - `function`: function object
  - `module`: collection of functions, variables, and other objects
  - `bool`: Boolean Values (`True` / `False`)
  - `None`: Null object (with value "None")

Python Type Tree



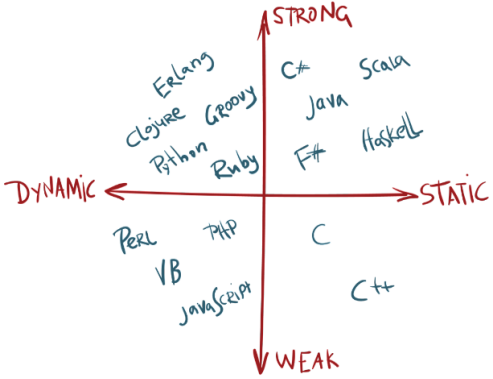
Benefits of Types

- **Operations**: Some operations between objects require the objects to be the same type
- **Comparisons**: Types allow for you to define how you can compare objects of different or same types
- **Builtin Functions**: Different types have different built in functions that let you things that make sense only with that type
- **Conversion Methods**: Each type may also have defined how to convert it to another type

Types in Programming Languages

C++: Can not compare float and int

Python:



Python Variables

- Variables are used to store values in a program.
- In Python, variables are created when you assign a value to them.
- Variables can be of **any type**, such as integers, floats, strings, and booleans.

Python Variables Examples

```
x = 5
y = 10
z = x + y
print(z) # this will print 15
```

Listing 1: Simple Variable Assignment

```
x = 5
print(x) # this will print 5
x = "Hello"
print(x) # this will print "Hello"
```

Listing 2: Reassigning Variables

Operators

- Operators are used to perform operations on variables and values.
- Python divides the operators in the following groups:
  - Arithmetic operators
  - Assignment operators
  - Comparison operators
  - Logical operators
  - Identity operators
  - Membership operators
  - Bitwise operators

Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

- `x + y`: Addition
- `x - y`: Subtraction
- `x * y`: Multiplication
- `x / y`: Division
- `x % y`: Modulus, remainder of x divided by y, only works with integers
- `x ** y`: Exponentiation, x to the power of y
- `x // y`: Floor Division, x divided by y rounded down to the nearest integer

Assignment Operators

Assignment operators are used to both compute and assign values to variables:

- `x = y`: Assignment, store the value of y in x
- `x += y`: Addition, add y to x and store the result in x
- `x -= y`: Subtraction, subtract y from x and store the result in x
- `x *= y`: Multiplication, multiply x by y and store the result in x
- `x /= y`: Division, divide x by y and store the result in x
- `x %= y`: Modulus, divide x by y and store the remainder in x
- `x **= y`: Exponentiation, raise x to the power of y and store the result in x
- `x //= y`: Floor Division, divide x by y, round down to the nearest integer, and store the result in x

Comparison Operators

Comparison operators are used to compare two values:

- `x == y`: Equality
- `x != y`: Inequality
- `x > y`: Greater than
- `x < y`: Less than
- `x >= y`: Greater than or equal to
- `x <= y`: Less than or equal to

Logical Operators

Logical operators are used to combine conditional statements:

- `x and y`: AND operator; if both statements are true, the result is true
- `x or y`: OR operator; if one of the statements is true, the result is true
- `not x`: NOT operator; returns the opposite of the value

Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

- `x is y`: Returns true if both variables are the same object instance
- `x is not y`: Returns true if both variables are not the same object instance

Membership Operators

Membership operators are used to test if a sequence is presented in an object:

- `x in y`: Check if object x is in sequence y
- `x not in y`: Check if object x is not in sequence y

Bitwise Operators

Bitwise operators are used to compare binary values:

- `x << y`: Shift bits in x to the left by y places, same as `x * (2 ** y)`
- `x >> y`: Shift bits in x to the right by y places, same as `x / (2 ** y)`
- `x & y`: Bitwise AND
- `x | y`: Bitwise OR
- $\sim$ `x`: Bitwise NOT
- `x ^ y`: Bitwise XOR, if both bits are the same, the result is 0, otherwise the result is 1.

Control Flow

- Control flow statements are used to change the execution order of the program.
- Python supports the following control flow statements:
  - `if` statement
  - `else` statement
  - `elif` statement

If Example

```
1 x = 5
2 y = 10
3 if x > y:
4     print("x is greater than y")
```

Listing 3: If Statement

- What will happen when we run this code?
- What will happen when we change the value of x to 10?

If-Else Example

```
1 x = 5
2 y = 10
3 if x > y:
4     print("x is greater than y")
5 else:
6     print("x is less than or equal to y")
```

Listing 4: If-Else Statement

- What will happen when we run this code?
- What will happen when we change the value of x to 10?

If-Elif-Else Example

```
1  x = 5
2  y = 10
3  if x > y:
4      print("x is greater than y")
5  elif x == y:
6      print("x is equal to y")
7  elif x < y:
8      print("x is less than y")
9  else:
10     print("How did we get here?")
```

Listing 5: If-Elif-Else Statement

- What will happen when we run this code?
- What will happen when we change the value of x to 10?
- What will happen when we change the value of x to 15?
- Can this code ever reach the `else` statement?
- What will happen when we change the value of x to "hello"?

Loops

- Loops are used to repeat a block of code.
- Python supports common patterns of loops:
  - `while` loop
  - `for` loop

While Loop Example

```
1 x = 0
2 while x < 5:
3     print(x)
4     x += 1
```

Listing 6: Simple While Loop

- What will happen when we run this code?
- What will happen if we change the value of x to 2?
- What will happen if we change the value of x to 10?

Complex While Loop Example

```
1  rate_in = 10
2  rate_out = 5
3  max_capacity = 100
4  current_capacity = 0
5  count = 0
6  while current_capacity < max_capacity:
7      current_capacity += rate_in - rate_out
8      count += 1
9
10 print(f"It took {count} minutes to fill the tank to max capacity.")
```

Listing 7: Complex While Loop

- What will happen when we run this code?
- What will happen if we change the value of rate_out to 10?
- What will happen if we change the value of rate_out to 15?
- What will happen if we change the value of rate_in to 500?

For Loop Example

```
1 my_list = ["Exam 1", "HW 1", "HW 2", "Exam 2"]
2 for item in my_list:
3     print(item)
```

Listing 8: For Loop

- What will happen when we run this code?
- What will happen if we change the value of my_list to

?

For Loop with Enumerate Example

```
1 my_list = ["Exam 1", "HW 1", "HW 2", "Exam 2"]
2 for i, item in enumerate(my_list):
3     print(f"{i}: {item}")
```

Listing 9: For Loop with Enumerate

- What will happen when we run this code?

For Loop with Range Example

```
1 for i in range(20):
2     print(i*5)
```

Listing 10: For Loop with Range

- What will happen when we run this code?
- What will happen if we change the value of range to 10?

Functions

- Functions are used to group a set of statements together to perform a specific task.
- Functions are defined using the `def` keyword.
- Functions can take arguments and return values.

Function Example 1

```
1 def my_function():
2     print("Hello World!")
3
4 my_function()
5 my_function()
```

Listing 11: Function Example 1

- What will happen when we run this code?

Function Example 2

```
1 def my_function(name):
2     print(f"Hello {name}!")
3
4 my_function("John")
5 my_function("Jane")
```

Listing 12: Function Example 2

- What will happen when we run this code?

Function Example 3

```
1 def my_function(name):
2     return f"Hello {name}!"
3
4 my_string = my_function("John")
5 print(my_string)
```

Listing 13: Function Example 3

- What will happen when we run this code?
- what will happen if I just call `my_function("John")`?

Table of Contents

Other Topics

- Python has many advanced concepts that we will not cover in this course.
- Some of these topics include:
  - `lambda` and `partial` functions
  - `list`, `dict`, and `set` detials and comprehension
  - `map`, `filter`, and `reduce` functions
  - `zip` and `enumerate` functions
  - builtin functions for different types
  - `class` and object oriented programming
  - `import` for importing local modules and from libraries
- These topics should be be covered in the supplemental python courses
- Some topics are also covered in the advanced python lecture
- **You still need to know how to use these topics for the rest of the course! Please make sure to review these topics on your own.**

Table of Contents

Python Standard Library

- The Python Standard Library is a set of modules that come with Python.
- These modules provide a wide range of functionality.
- Some of the most commonly used modules include:
  - `re`: Regular expression operations
  - `math`: Mathematical functions
  - `random`: Generate pseudo-random numbers
  - `itertools`: Functions creating iterators for efficient looping
  - `functools`: Higher-order functions and operations on callable objects
  - `csv`: CSV File Reading and Writing
  - `os`: Miscellaneous operating system interfaces
  - `datetime`: Basic date and time types and manipulations
  - `argparse`: Parser for command-line options, arguments and sub-commands
  - `json`: JSON encoder and decoder
  - `multiprocessing`: Process-based parallelism
- Ther are also many more modules that are not listed here but are really useful.
- Some of these are also covered in the advanced python lecture.

Table of Contents

Python Packages

- Python packages are a way of organizing and distributing Python code.
- Lets use code someone else wrote in your own code.
- Install packages using `pip` or `conda` from a package repository.
- Once installed, you can import the functions and class from the package to use in your code.

Example Packages

- **NumPy**: Fundamental package for scientific computing based on arrays
- **Matplotlib**: Comprehensive library for creating static, animated, and interactive visualizations
- **Seaborn**: Data visualization library based on matplotlib with a simpler high-level interface
- **SciPy**: Collection user-friendly and efficient numerical routines, such as routines for numerical integration, interpolation, optimization, linear algebra, statistics, and signal processing
- **Pandas**: Data analysis and manipulation library based around tabular data
- **Scikit-Image**: Image processing toolbox
- **Scikit-Learn**: A machine learning toolbox
- **PyTorch**: A comprehensive framework for deep learning

Table of Contents

Conclusion

- You will be using Python for the rest of the course.
- Address **all** you Python questions and confusion now so that you can focus on the material as we progress.
- Some more complex topics will be covered in the advanced python lecture which is also highly recommended to make your life easier when writing code for research.