ALGONQUIN COLLEGE

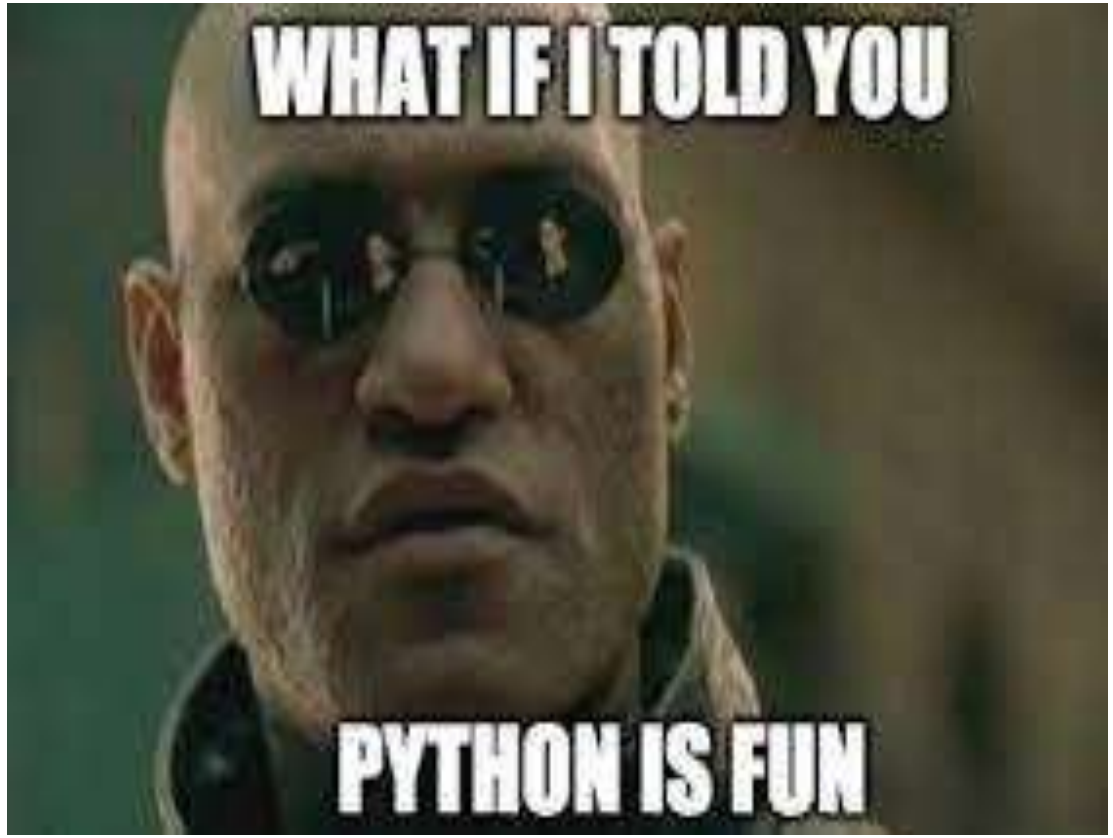
NET2008 DEVOPS

INTRO TO PYTHON

Introduction to Python

Background
Data Types/Structure
Control flow
Functions
Classes and Objects
Modules and Packages
File I/O

Life Is Short, Use Python



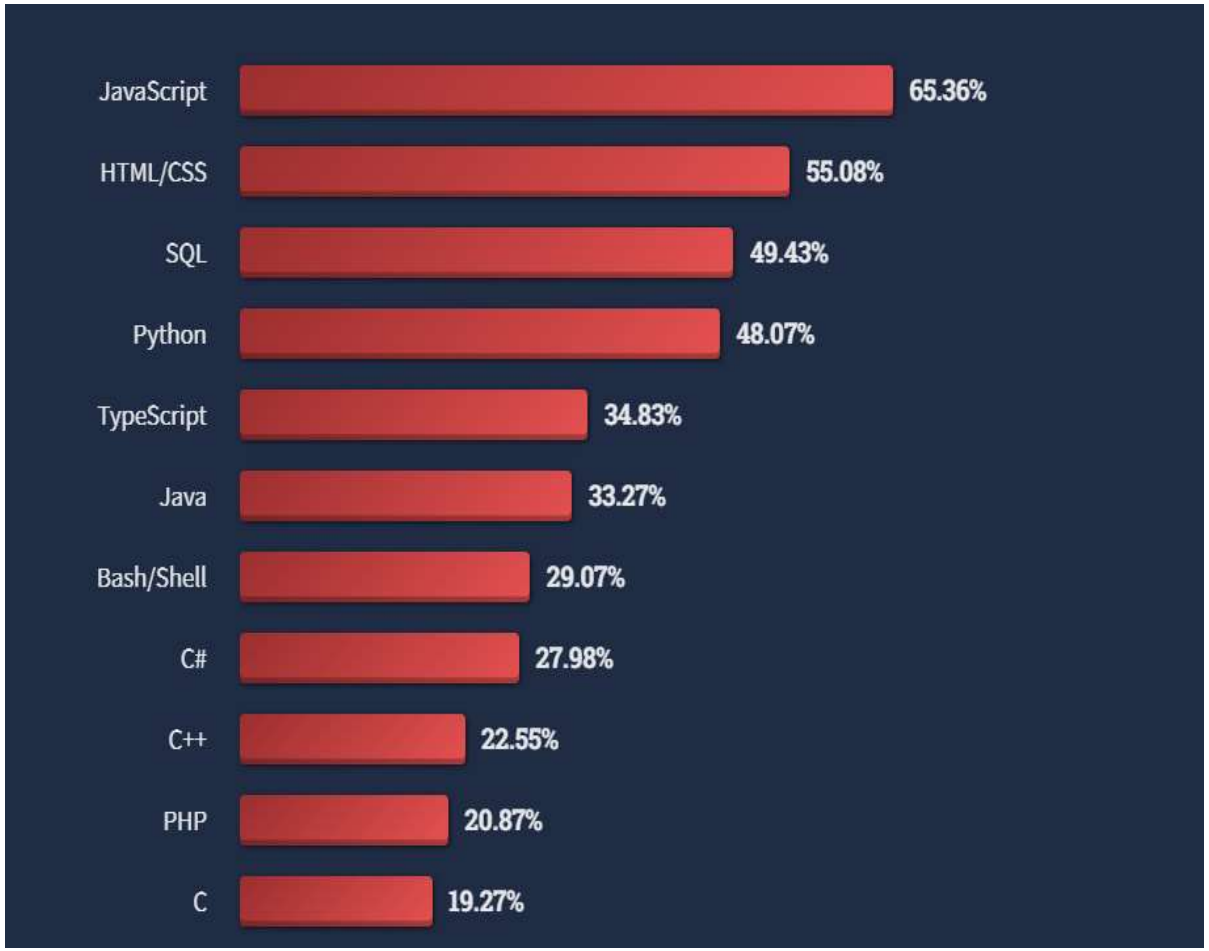
For More Information About Python

- Official Site:

  https://www.python.org/
- Books:

  https://wiki.python.org/moin/PythonBooks

Most Popular Languages (2022)



Source:

Stack Overflow 2022 Survey

Running the first Python program

- Python programs are executed by an interpreter, which means the code is fed through this interpreter to be executed by the underlying operating system and results are displayed.
  - Interactive mode
  - Run Python file

```
C:\Users\vasve>python3
Python 3.8.10 (tags/v3.8.10:3d8993a, May  3 2021, 11:48:03) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("NET2008")
NET2008
>>>
```

Python Interactive Shell

You can type things directly into a running Python session

```
C:\Users\vasve>python3
Python 3.8.10 (tags/v3.8.10:3d8993a, May  3 2021, 11:48:03) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 1+2
3
>>> name = "NET2008"
>>> name
'NET2008'
>>> print("Class ",name)
Class  NET2008
>>>
```

Python Versions

- Python has been going through a transition from Python 2 to Python 4 for the last few years.
- Python 3 was released back in 2008, over 10 years ago, with active development with the most recent release of 3.10
- Unfortunately, Python 3 is not backward compatible with Python 2.

Introduction to Python

Background
Data Types/Structure
Control flow
Functions
Classes and Objects
Modules and Packages
File I/O

Python Built-in Types

Python has several standard types built into the interpreter:

- **None**: The Null object
- **Numerics**: int, long, float, complex, and bool (the subclass of int with a True or False value)
- **Sequences**: str, list, tuple
- **Mappings**: dict
- **Sets**: set

None Type

- The **None** type denotes an object with no value.
- The **None** type is returned in functions that do not explicitly return anything.
- The **None** type is also used in function arguments to error out if the caller does not pass in an actual value.

Numeric

- Python numeric objects are basically numbers.
- Except for Booleans, the numeric types of int, long, float, and complex are all signed, meaning they can be positive or negative.
- A Boolean is a subclass of the integer, which can be one of two values: 1 for True, and 0 for False.
- The rest of the numeric types are differentiated by how precisely they can represent the number; for example, int are whole numbers with a limited range while long are whole numbers with unlimited range.
- Floats are numbers using the double precision representation (64-bit) on the machine.

Sequences - String

- Sometimes it surprises people that string is actually a sequence type. But if you look closely, strings are a series of characters put together. Strings are enclosed by either single, double, or triple quotes.
- Note in the following examples, the quotes must match, and triple quotes allow the string to span different lines:

```
a = "networking is fun"
b = 'DevOps is fun too'
c = """what about coding?
super fun!"""

print("value of a: ", a)
print(b)
print(c)
```

Sequences - List

- The two commonly used sequence types are **lists** and **tuples**. Lists are sequences of arbitrary objects.
- Lists can be created by enclosing objects in square brackets. Just like strings, lists are indexed by non-zero integers that start at zero. The values of lists are retrieved by referencing the index number:

```
vendors = ["Cisco", "Arista", "Juniper"]
print(vendors[0])
```

Strings share many features with lists

```
>>> smiles = "C(=N)(N)N.C(=O)(O)O"
>>> smiles[0]
'C'
>>> smiles[1]
'('
>>> smiles[-1]
'O'
>>> smiles[1:5]
'(=N)'
>>> smiles[10:-4]
'C(=O)'
```

Use “slice” notation to get a substring

String Methods: find, split

```
smiles = "C(=N)(N)N.C(=O)(O)O"
smiles.find("(O)")
15

smiles.find(".")
9

smiles.find(".", 10)
-1

smiles.split(".")
['C(=N)(N)N', 'C(=O)(O)O']
```

Use “find” to find the start of a substring.

Start looking at position 10.

Find returns -1 if it couldn’t find a match.

Split the string into parts with “.” as the delimiter

String operators: in, not in

```
if "Br" in "Brother":
   print("contains brother")
contains brother

email_address = ”bill"
if "@" not in email_address:
   email_address += "@carleton.ca"

print(email_address)
bill@carleton.ca
```

More String methods

```
email = "bill@carleton.ca"
email.startswith("b")
True
email.endswith("u")
False

"%s@carleton.ca" % "bill"
'bill@carleton.ca'

names = ['NET', '2008', 'A']
", ".join(names)
'NET, 2008, A'

"bill".upper()
'BILL'
```

Unexpected things about strings

```
>>> s = "andrew"
>>> s[0] = "A"
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item
  assignment
>>> s = "A" + s[1:]
>>> s
'Andrew`
```

Strings are read only

"\" is for special characters

```
\n  -> newline
\t  -> tab
\\  -> backslash
...
```

Lists are mutable - some useful methods

```
>>> ids = ["9pti", "2plv", "1crn"]
>>> ids.append("1alm")
>>> ids
['9pti', '2plv', '1crn', '1alm']
>>>ids.extend(L)
    Extend the list by appending all the items
    in the given list; equivalent to a[len(a):] = L.
>>> del ids[0]
>>> ids
['2plv', '1crn', '1alm']
>>> ids.sort()
>>> ids
['1alm', '1crn', '2plv']
>>> ids.reverse()
>>> ids
['2plv', '1crn', '1alm']
>>> ids.insert(0, "9pti")
>>> ids
['9pti', '2plv', '1crn', '1alm']
```

append an element

remove an element

sort by default order

reverse the elements in a list

insert an element at some specified position.
(Slower than .append())

Zipping lists together

```
>>> names = ['Yan', 'Sarah', 'Rick’]

>>> levels = [1, 2, 2]

>>> list(zip(names, levels))
[(‘Yan’, 1), (‘Sarah’, 2), (‘Rick’, 2)]
```