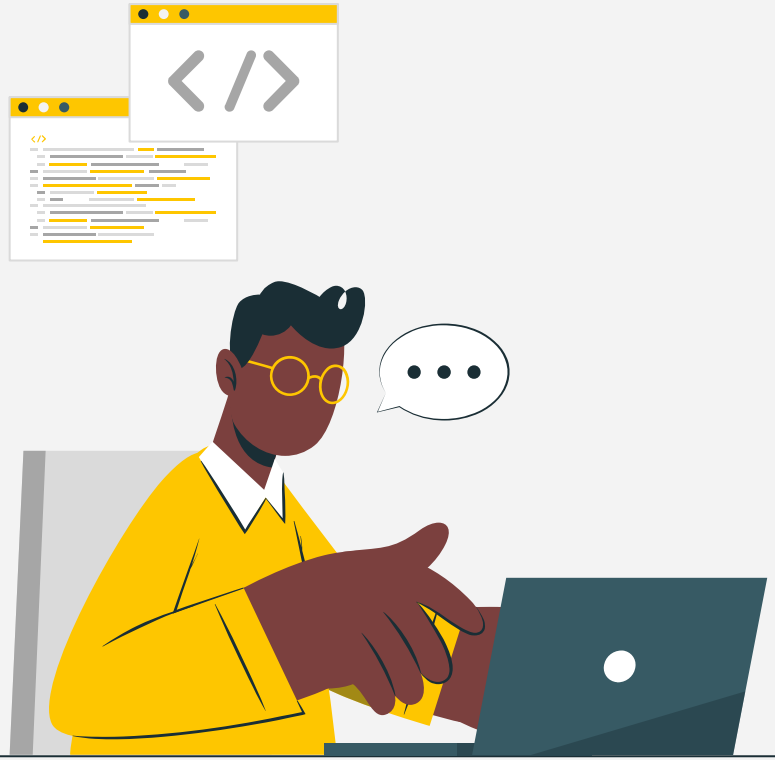
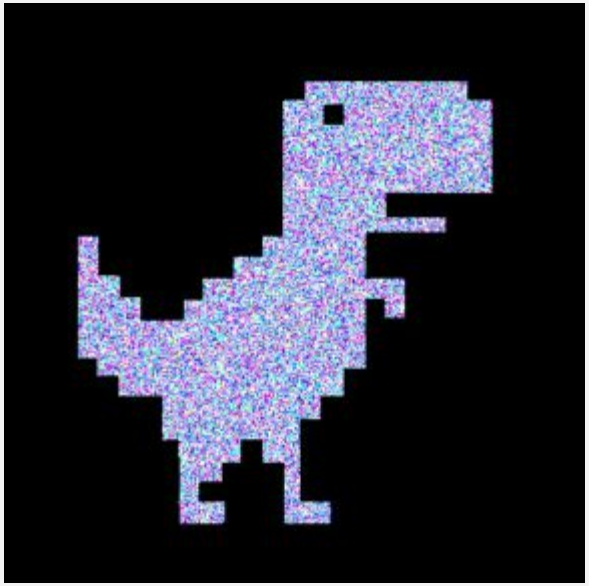


Python is Compiled!

(And How to Analyze its Bytecode)



Hi! I'm Ivy

they/them - 2A ~~SYDE~~ CS
A.K.A hexadecimal_dinosaur
Forensics + RevEng @ sillysec (https://cve.gay/)
Stores too much malware on their computer

Python is Compiled?

Compiled

Source code is converted into a binary with a compiler and linker

- C/C++
- Rust
- Go
- Haskell

Interpreted

Code is parsed and executed line-by-line by an interpreter

- Bash
- PowerShell
- JavaScript
- Perl

Bytecode Languages

Binary Compiled

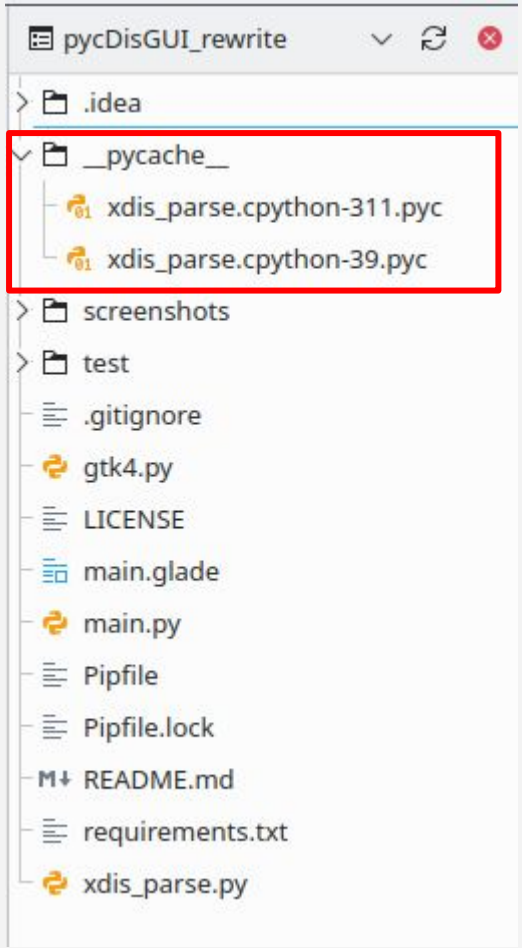
- C/C++
- Rust
- Go
- Haskell

Bytecode Compiled

- Java (to .class files)
- Python (to .pyc files)

Interpreted

- Bash
- PowerShell
- JavaScript
- Perl



.pyc Files

Compiled at runtime
Contains Python code objects
Cached in the `__pycache__` folder

```
In [2]: def print_hello(name: str) -> None:
   ...:     print("Hello " + name + "!")
   ...:

In [3]: code = print_hello.__code__

In [4]: type(code)
Out[4]: code

In [5]: code
Out[5]: <code object print_hello at 0x79b49c26fd20, file "/tmp/ipykernel_12333/398923408.py", line 1>

In [6]: code.co_consts
Out[6]: (None, 'Hello ', '!')

In [7]: code.co_names
Out[7]: ('print',)

In [8]: code.co_argcount
Out[8]: 1

In [9]: code.co_varnames
Out[9]: ('name',)
```

```
import dis

code = compile(open("misc_code.py", "r").read(), "misc_code.py", "exec")

print(code)

dis.disassemble(code)
```

Python Bytecode

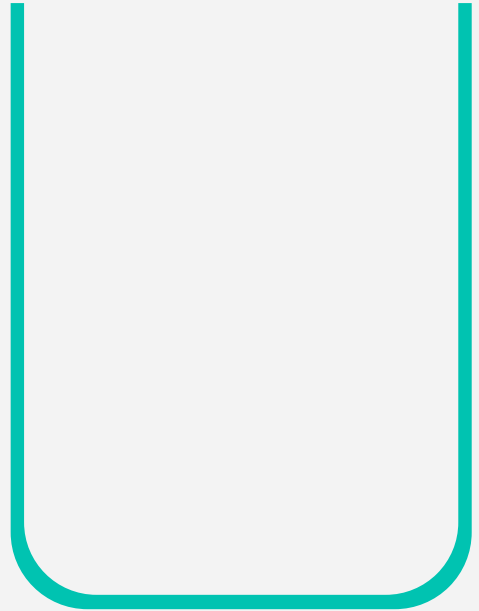
```
In [14]: import dis

In [15]: dis.disassemble(code)
  1           0 RESUME                   0

  2           2 LOAD_GLOBAL              1 (NULL + print)
             14 LOAD_CONST               1 ('Hello ')
             16 LOAD_FAST                0 (name)
             18 BINARY_OP                0 (+)
             22 LOAD_CONST               2 ('!')
             24 BINARY_OP                0 (+)
             28 PRECALL                  1
             32 CALL                     1
             42 POP_TOP
             44 LOAD_CONST               0 (None)
             46 RETURN_VALUE
```

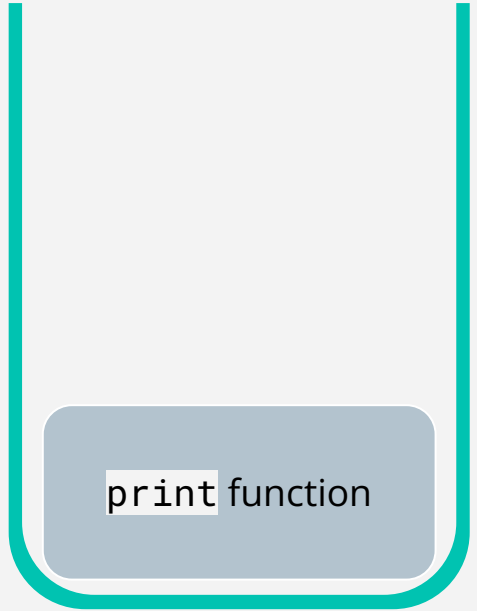
Stack Based Instruction Set?

```
2       2       LOAD_GLOBAL      1 (NULL + print)
        14      LOAD_CONST       1 ('Hello ')
        16      LOAD_FAST        0 (name)
        18      BINARY_OP        0 (+)
        22      LOAD_CONST       2 ('!')
        24      BINARY_OP        0 (+)
        28      PRECALL          1
        32      CALL             1
        42      POP_TOP
        44      LOAD_CONST        0 (None)
        46      RETURN_VALUE
```



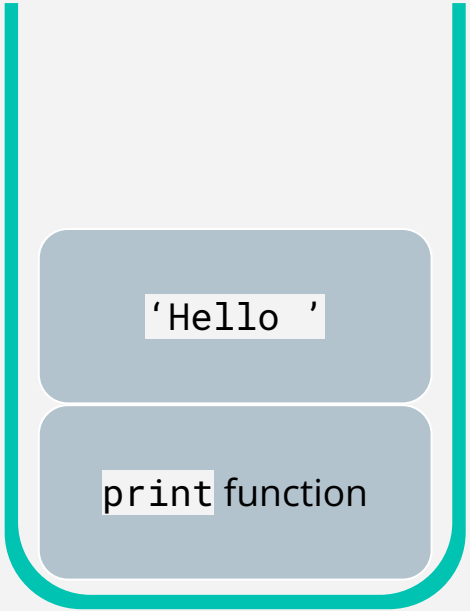
Stack Based Instruction Set?

```
2       2       LOAD_GLOBAL      1 (NULL + print)
        14      LOAD_CONST       1 ('Hello ')
        16      LOAD_FAST        0 (name)
        18      BINARY_OP        0 (+)
        22      LOAD_CONST       2 ('!')
        24      BINARY_OP        0 (+)
        28      PRECALL          1
        32      CALL             1
        42      POP_TOP
        44      LOAD_CONST       0 (None)
        46      RETURN_VALUE
```



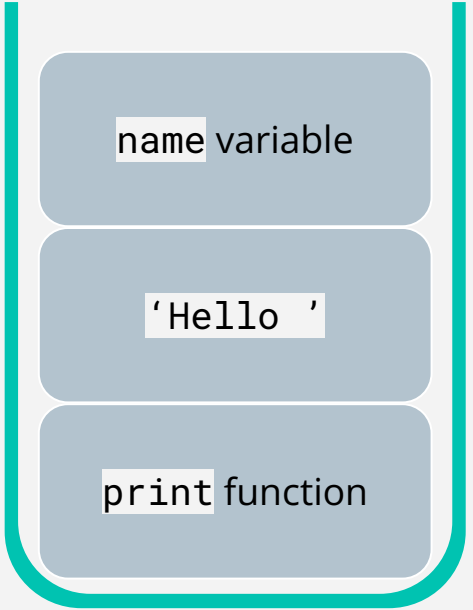
Stack Based Instruction Set?

```
2       2       LOAD_GLOBAL      1 (NULL + print)
        14      LOAD_CONST       1 ('Hello ')
        16      LOAD_FAST        0 (name)
        18      BINARY_OP        0 (+)
        22      LOAD_CONST       2 ('!')
        24      BINARY_OP        0 (+)
        28      PRECALL          1
        32      CALL             1
        42      POP_TOP
        44      LOAD_CONST       0 (None)
        46      RETURN_VALUE
```



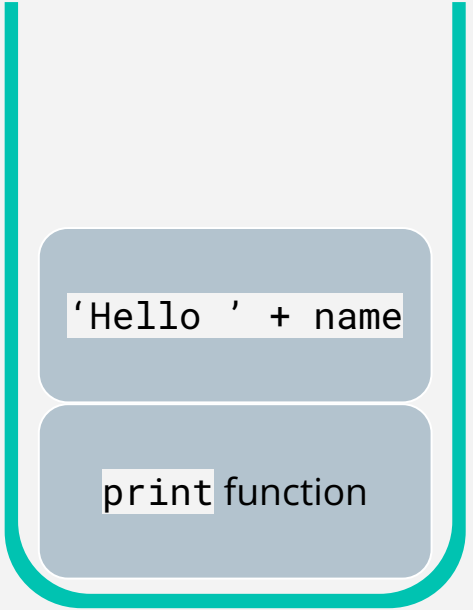
Stack Based Instruction Set?

```
2       2       LOAD_GLOBAL      1 (NULL + print)
        14      LOAD_CONST       1 ('Hello ')
        16      LOAD_FAST        0 (name)
        18      BINARY_OP        0 (+)
        22      LOAD_CONST       2 ('!')
        24      BINARY_OP        0 (+)
        28      PRECALL          1
        32      CALL             1
        42      POP_TOP
        44      LOAD_CONST       0 (None)
        46      RETURN_VALUE
```



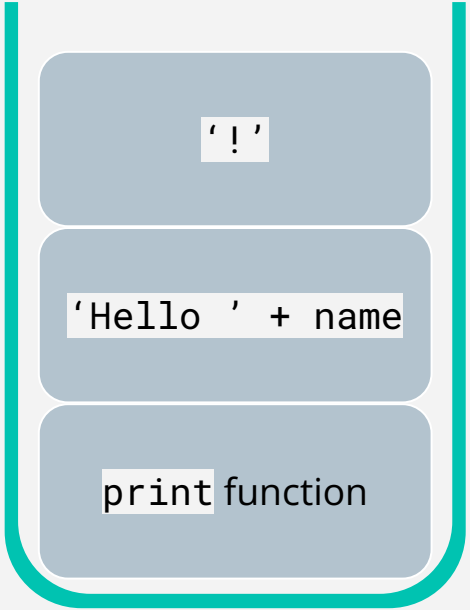
Stack Based Instruction Set?

```
2       2       LOAD_GLOBAL      1 (NULL + print)
        14      LOAD_CONST       1 ('Hello ')
        16      LOAD_FAST        0 (name)
        18      BINARY_OP        0 (+)
        22      LOAD_CONST       2 ('!')
        24      BINARY_OP        0 (+)
        28      PRECALL          1
        32      CALL             1
        42      POP_TOP
        44      LOAD_CONST        0 (None)
        46      RETURN_VALUE
```



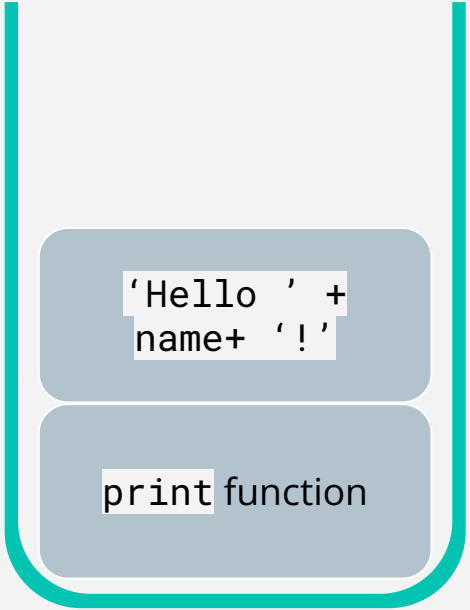
Stack Based Instruction Set?

```
2       2       LOAD_GLOBAL      1 (NULL + print)
        14      LOAD_CONST       1 ('Hello ')
        16      LOAD_FAST        0 (name)
        18      BINARY_OP        0 (+)
        22      LOAD_CONST       2 ('!')
        24      BINARY_OP        0 (+)
        28      PRECALL          1
        32      CALL             1
        42      POP_TOP
        44      LOAD_CONST       0 (None)
        46      RETURN_VALUE
```



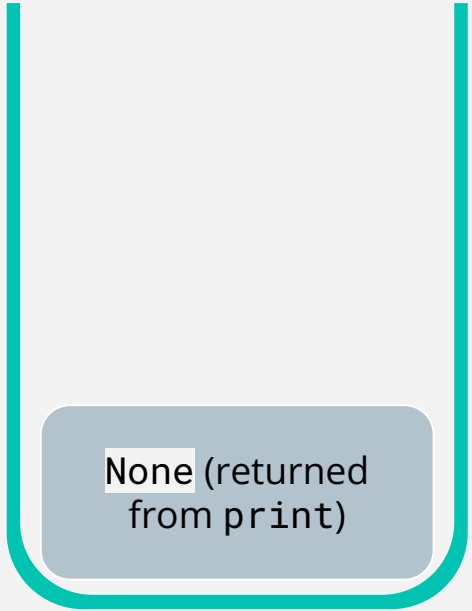
Stack Based Instruction Set?

```
2       2       LOAD_GLOBAL      1 (NULL + print)
        14      LOAD_CONST       1 ('Hello ')
        16      LOAD_FAST        0 (name)
        18      BINARY_OP        0 (+)
        22      LOAD_CONST       2('!')
        24      BINARY_OP        0 (+)
        28      PRECALL          1
        32      CALL             1
        42      POP_TOP
        44      LOAD_CONST       0 (None)
        46      RETURN_VALUE
```



Stack Based Instruction Set?

```
2       2       LOAD_GLOBAL      1 (NULL + print)
        14      LOAD_CONST       1 ('Hello ')
        16      LOAD_FAST        0 (name)
        18      BINARY_OP        0 (+)
        22      LOAD_CONST       2 ('!')
        24      BINARY_OP        0 (+)
        28      PRECALL          1
        32      CALL             1
        42      POP_TOP
        44      LOAD_CONST       0 (None)
        46      RETURN_VALUE
```




```
print("Hello " + name + "!")
```

Demo???

How is this useful?

Resources

- Python dis module documentation - https://docs.python.org/3/library/dis.html
- Inside the Python Virtual Machine (Obi Ike-Nwosu) - https://leanpub.com/insidethepythonvirtualmachine/read
- Python's Innards (Yaniv Aknin) - https://tech.blog.aknin.name/category/my-projects/pythons-innards/
- Is Python interpreted or compiled? Yes. (Ned Batchelder) - https://nedbatchelder.com/blog/201803/is_python_interpreted_or_compiled_yes.html
- xdis Python disassembler - https://github.com/rocky/python-xdis