The Journey Of A Program

ASSOCIATION FOR COMPUTING MACHINERY AT UAB PRESENT:

LEETCODE NIGHT

SOLVE LEETCODE PROBLEMS TOGETHER WITH PEERS AND ENHANCE YOUR PROGRAMMING AND PROBLEM SOLVING SKILLS!

WEDNESDAY, JANUARY 22
4:30PM - 5:30PM
UH-1008 : FIRST FLOOR UH

Lab 1

“Python is an interpreted language. The source code is executed line by line”

“Python is directly interpreted”

“As an interpreted language, python code executes directly meaning there is no need to have a separate step for compiling”

“Python is purely an interpreted language, not being compiled at all”

Do these work?

```
def x():
    y()
def y():
    print(“X”)
x()
```

```
#include <stdio.h>
main() {
    x();
}
x() {
    y();
}
y() {
    printf("X\n");
}
```

```
#include <iostream>
int main() {
    x();
    return 0;
}
void x() {
    y();
}
void y() {
    std::cout << "X" <<
std::endl;
}
```

Well, how does it get run?

Python/Java

- Lexical Analysis (Tokenization)
- Syntax Analysis (Parsing)
- Semantic Analysis
- Bytecode Generation
- Ran by the Virtual Machine
    - Java can continue on and get optimized by JIT compilation
    - JIT Compilation translates bytecode to machine code

C/C++

- Preprocessing (ie: including includes)
- Lexical Analysis (Tokenization)
- Syntax Analysis (Parsing)
- Optimization
- Code Generation (Assembly)
- Code Generation (Object files)
- Linking (multiple object files)
- Executable

Note

Python 3.13 includes a JIT compiler to allow you to compile Python to machine code <- more like Java

It is disabled by default. If you want to try it out:

```
python3 --enable-experimental-jit <filename>
```

Well, how does it get run?

Lexical Analysis (Tokenization)

~~~~
```python
 def x():
     y()
 def y():
     print(“X”)
 x()
```
~~~~

1	NAME	def	Defines a function
2	NAME	x	Function name
3	OP	(	Opening parenthesis
4	OP	)	Closing parenthesis
5	OP	:	Colon indicating the start of block
6	INDENT	Indent	Indentation for the function body
7	NAME	y	Function call to y

Compilation Process Step 3

Syntax Analysis (Abstract Syntax Tree) (Parsing)

```
Module(
    body=[
        FunctionDef(
            name='x',
            args=arguments(),
            body=[
                Expr(
                    value=Call(
                        func=Name(id='y', ctx=Load()))))]),
```

```
FunctionDef(
            name='y',
            args=arguments(),
            body=[
                Expr(
                    value=Call(
                        func=Name(id='print', ctx=Load()),
                        args=[
                            Constant(value='X')]))]),
        Expr(
            value=Call(
                func=Name(id='x', ctx=Load()))))])
```

Well, how does it get run?

Bytecode Generation (Absolutely gibberish, but this is the disassembled version)

```
Disassembly of x:

  1           RESUME            0

  2           LOAD_GLOBAL       1 (y + NULL)

              CALL              0

              POP_TOP

              RETURN_CONST      0 (None)
```

```
Disassembly of y:

  4           RESUME            0


  5           LOAD_GLOBAL       1 (print + NULL)

              LOAD_CONST        1 ('X')

              CALL              1

              POP_TOP

              RETURN_CONST      0 (None)
```

This is what is contained in a .pyc file

What about your lab 2

Checking if every opening parentheses has a closing parentheses

- Super useful for programming language design

  def x(), int main(), for (), etc. All have parentheses

- A small precursor to writing your own interpreters
- Let’s do a subset of the problem - in Python

Credits

First page image: https://in.pinterest.com/pin/830914200034007461/