# Creative Coding 9

*Dissecting the Serpent.*

David Lynch

Slide Deck by Ruán Murgatroyd

## Last Class

- Python Dictionaries.
- Introducing data structures (Linked Lists and Stacks).
- How to use and reason with these structures.
- Implementing *Snake* in Python.

## Today’s Class

- An overview of Python and how it’s used.
- ‘The Road to Hello World!’, how your PC handles the task of saying hi!
- How Python is implemented and an in-depth look at CPython.
- What is a ‘Compiler’, and a high-level view of compilation.
- Other implementations of Python.
- Implementing a Stack-Based virtual machine in Python.

## Just what *is* Python?

And using Python in the command line.

## Just what *is* Python?

Python is a *dynamically typed* language with *Strong Typing*. It supports many high-level concepts like first-class functions, classes and more.

**Dynamically Typed** – A variable can hold multiple types of data throughout the execution of a program.

**Strong Typing** – Operations between variables of different types being banned. (I.e. trying to add a string to an integer)

## Main Implementations of Python

The most popular Python implementation is *CPython*. This is a *bytecode* interpreter and is a *Stack Based Virtual Machine*.

**Interpreter** – A program which executes source code line-by-line.

**Bytecode** – A binary instruction format used by an interpreter to execute code.

**Virtual Machine** – A piece of software which pretends to be a computer and is convinced it is one.

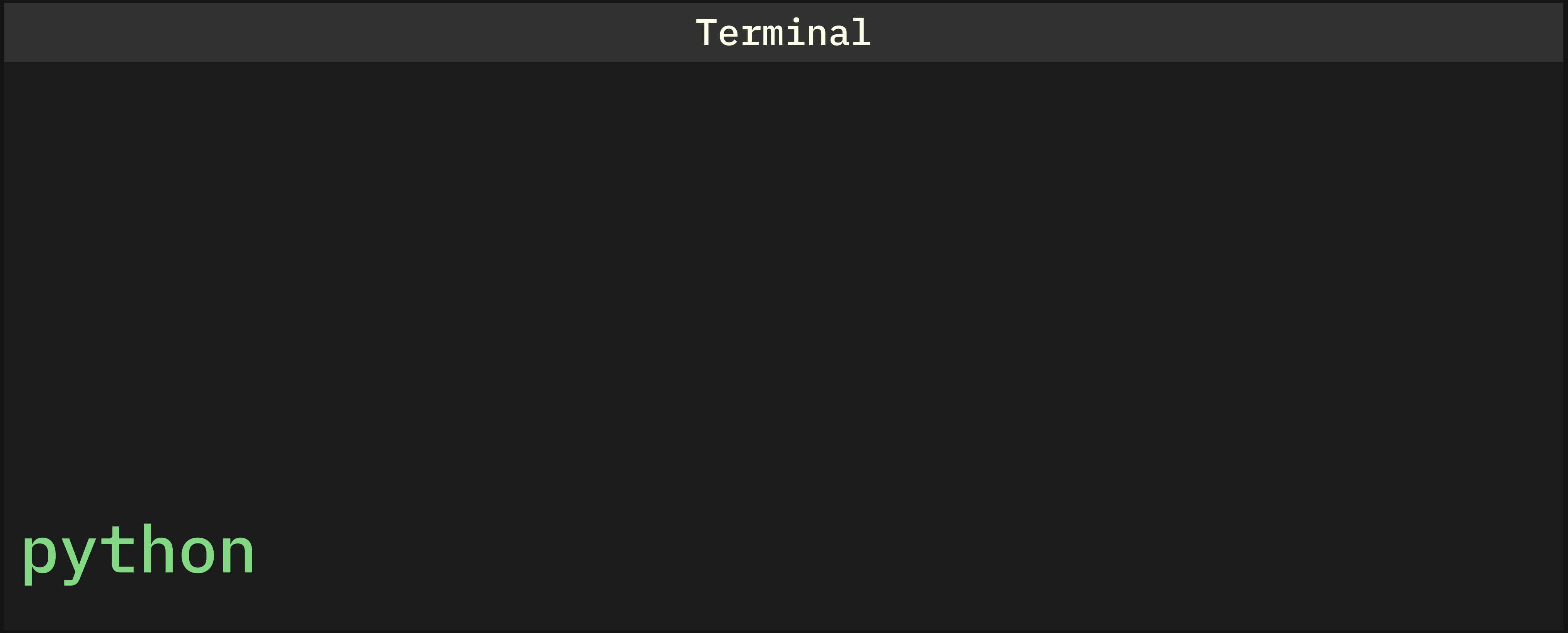
**Stack Machine** – A virtual or physical machine in which the primary form of data storage is a *stack.* It's important to note that stack machines are **Turing complete**.

## Python in the Command Line

Python is a command-line application. This means its main usage is through typing commands into your terminal.

# Python in the Command Line

Python is a command-line application. This means its main usage is through typing commands into your terminal.

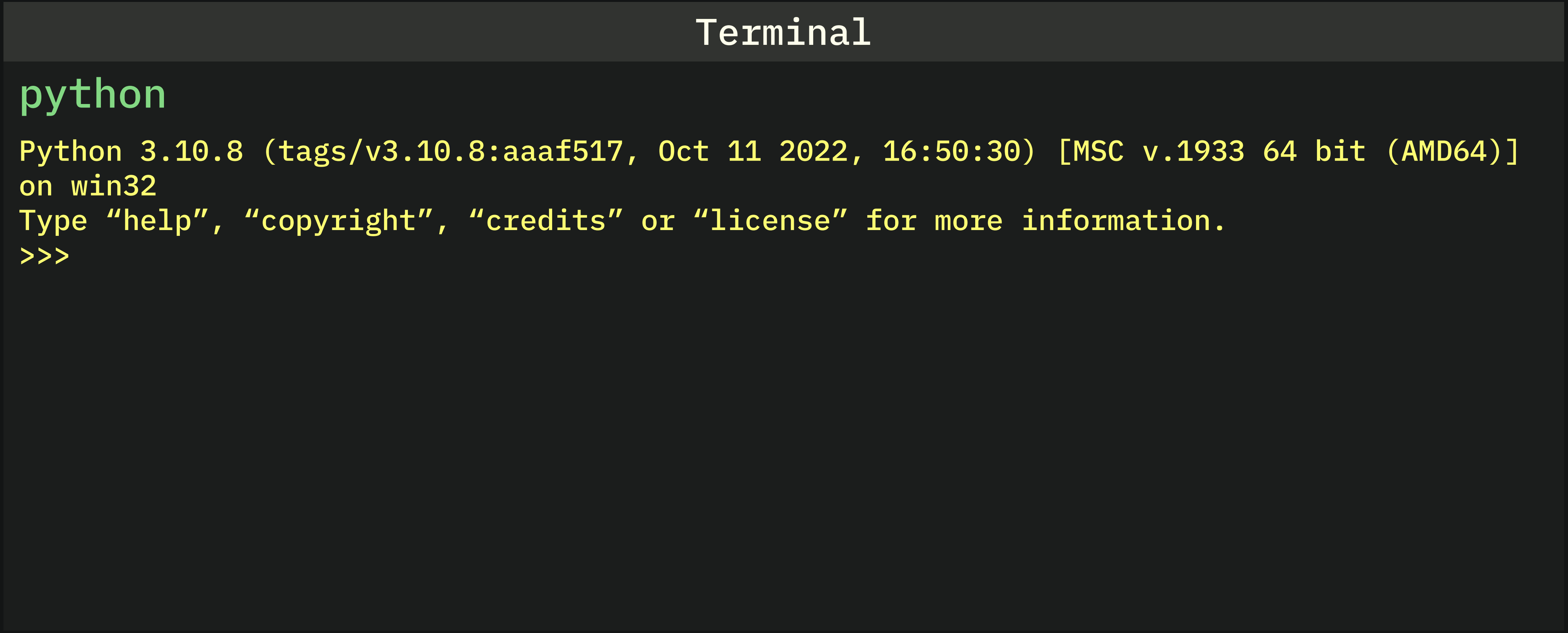


```
python
```

# Python in the Command Line

Python is a command-line application. This means its main usage is through typing commands into your terminal.

This loads you into an interactive session.



Terminal

```
python
Python 3.10.8 (tags/v3.10.8:aaaf517, Oct 11 2022, 16:50:30) [MSC v.1933 64 bit (AMD64)]
on win32
Type “help”, “copyright”, “credits” or “license” for more information.
>>>
```

## Python in the Command Line

Python is a command-line application. This means its main usage is through typing commands into your terminal.

Terminal

```
python

Python 3.10.8 (tags/v3.10.8:aaaf517, Oct 11 2022, 16:50:30) [MSC v.1933 64 bit (AMD64)]
on win32
Type “help”, “copyright”, “credits” or “license” for more information.
>>> exit()






python hello.py
```

# Python in the Command Line

Python is a command-line application. This means its main usage is through typing commands into your terminal.

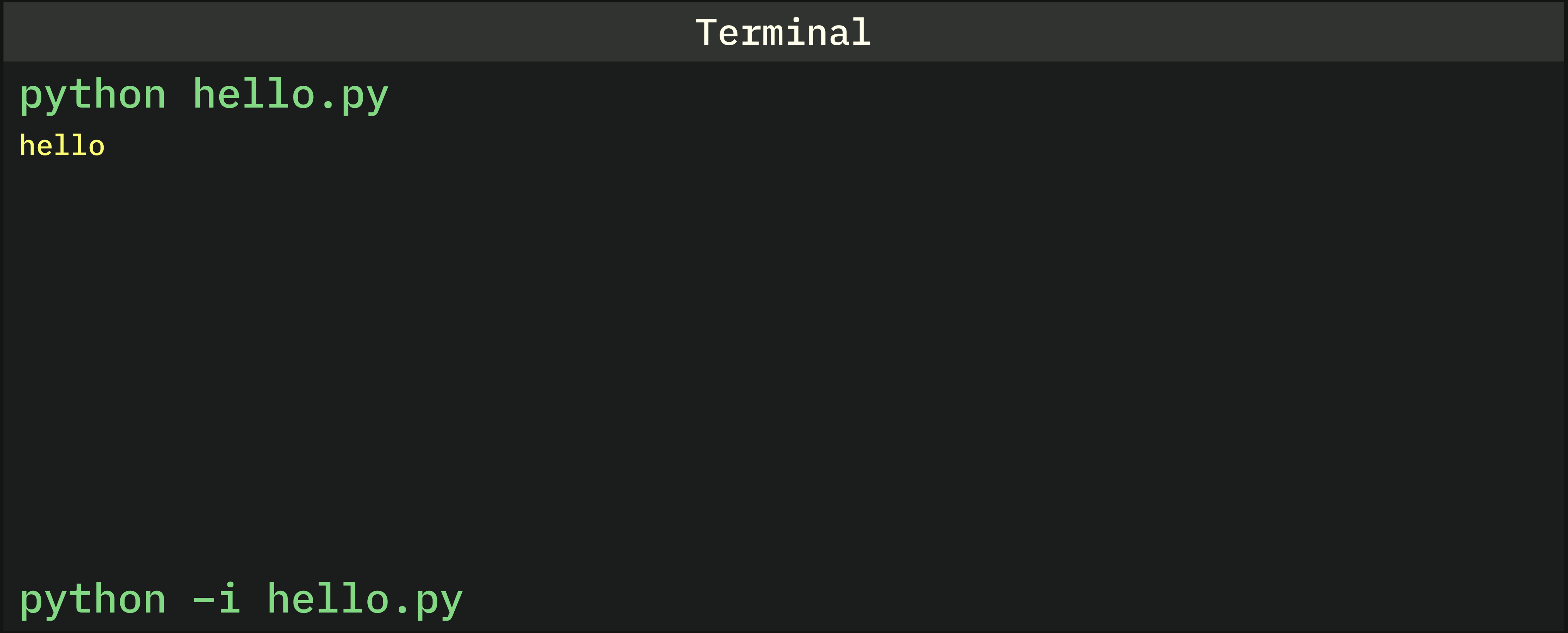
Terminal

```
python
Python 3.10.8 (tags/v3.10.8:aaaf517, Oct 11 2022, 16:50:30) [MSC v.1933 64 bit (AMD64)]
on win32
Type “help”, “copyright”, “credits” or “license” for more information.
>>> exit()
python hello.py
hello
```

Runs a python script with that name and exits.

# Python in the Command Line

Python is a command-line application. This means its main usage is through typing commands into your terminal.



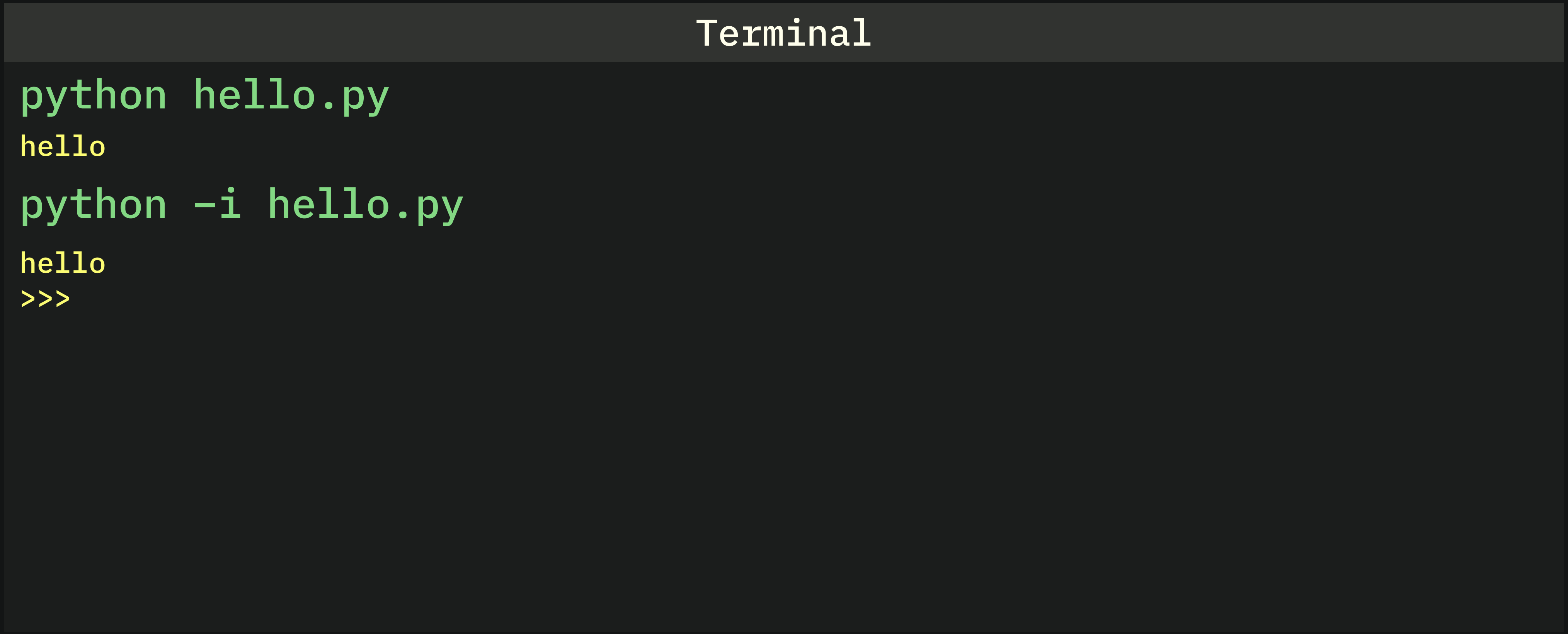
Terminal

```
python hello.py
hello




python -i hello.py
```

# Python in the Command Line

Python is a command-line application. This means its main usage is through typing commands into your terminal.

Runs a python script with that name and then puts you in an interactive session



Terminal

```
python hello.py
hello
python -i hello.py
hello
>>>
```

# Python in the Command Line

Python is a command-line application. This means its main usage is through typing commands into your terminal.

Terminal

```
python -i hello.py
hello
>>> exit()

python -c print(“hi”)
```

# Python in the Command Line

Python is a command-line application. This means its main usage is through typing commands into your terminal.

Terminal

```
python -i hello.py
hello
>>> exit()
python -c “print(“hi”)”
hi
```

Passes a string from the command line in as code and executes it.

## Python in the Command Line

Python has many other flags that can be passed to Python. But -i and –c are the most important.

| | | | |
|---|---|---|---|
| -c | -d | -P | -v |
| -m | -E | -q | -W |
| -h | -i | -R | -x |
| -v | -I | -s | -X |
| -b | -O | -S | -J |
| -B | -OO | -u | <script> |

## The Road to ‘Hello World’

## The Road to ‘Hello World’

In Python we just need to type `print("Hello World")` to write ‘Hello World’ to our screen. Passing this text directly to the CPU could cause data corruption, and to understand how this is printed, we will look at:

1. Python’s Hello World Program
2. C’s Hello World Program
3. Assembly’s Hello World Program

We will be stepping through the commands needed to run these programs as well as what they’re doing.

All examples will be done with an x86_64 CPU on a Linux Operating System with glibc.

## The Road to ‘Hello World’

helloWorld.py

```python
print("Hello World")
```

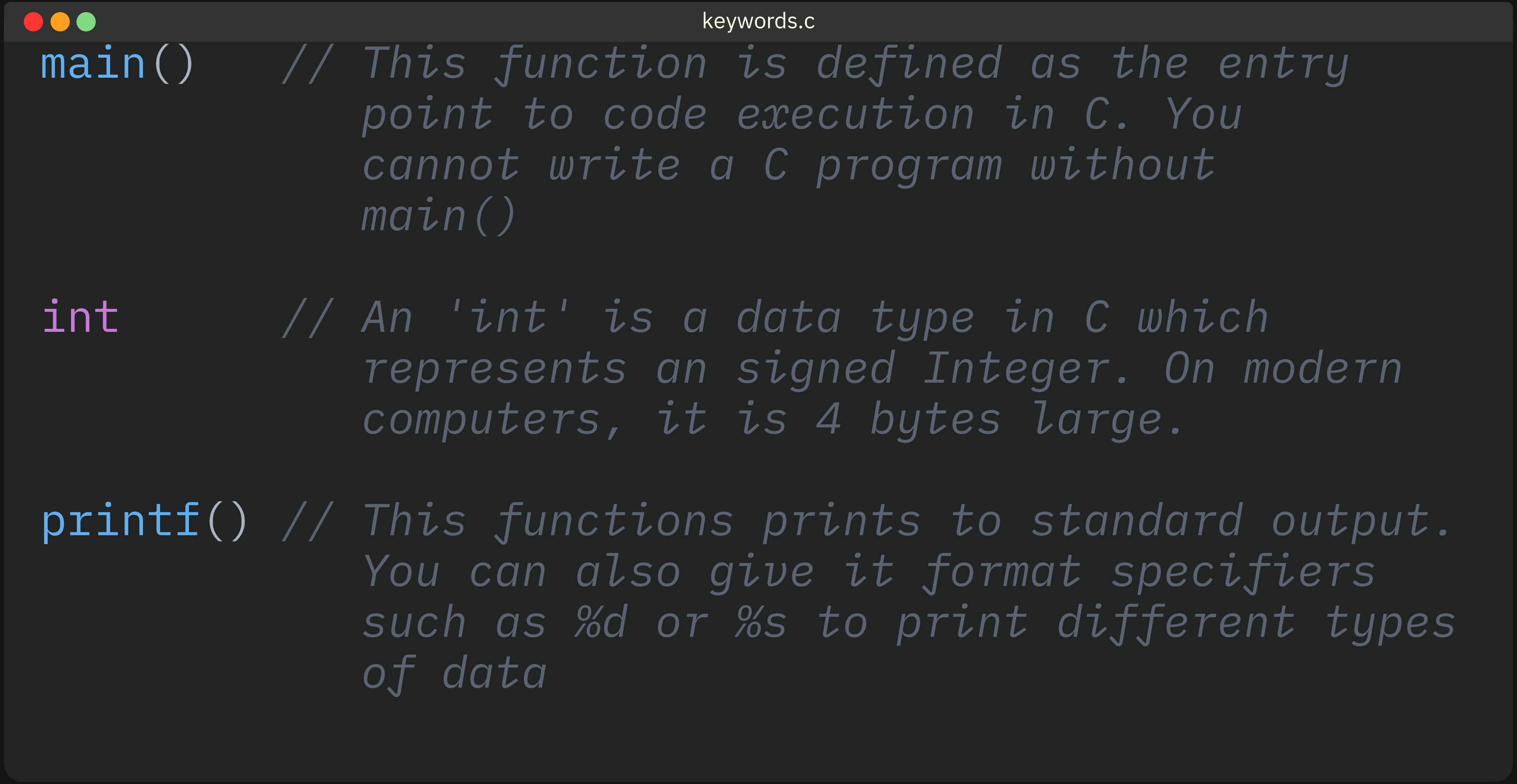
helloWorld.c

```c
#include <stdio.h>

int main()
{

  printf("Hello World!\n");

  return 0;
}
```

helloWorld.asm

```asm
global _start

section .text

_start:
  mov rax, 1
  mov rdi, 1
  mov rsi, msg
  mov rdx, msglen
  syscall

  mov rax, 60
  mov rdi, 0
  syscall

section .rodata
  msg: db "Hello, world!", 10
  msglen: equ $ - msg
```

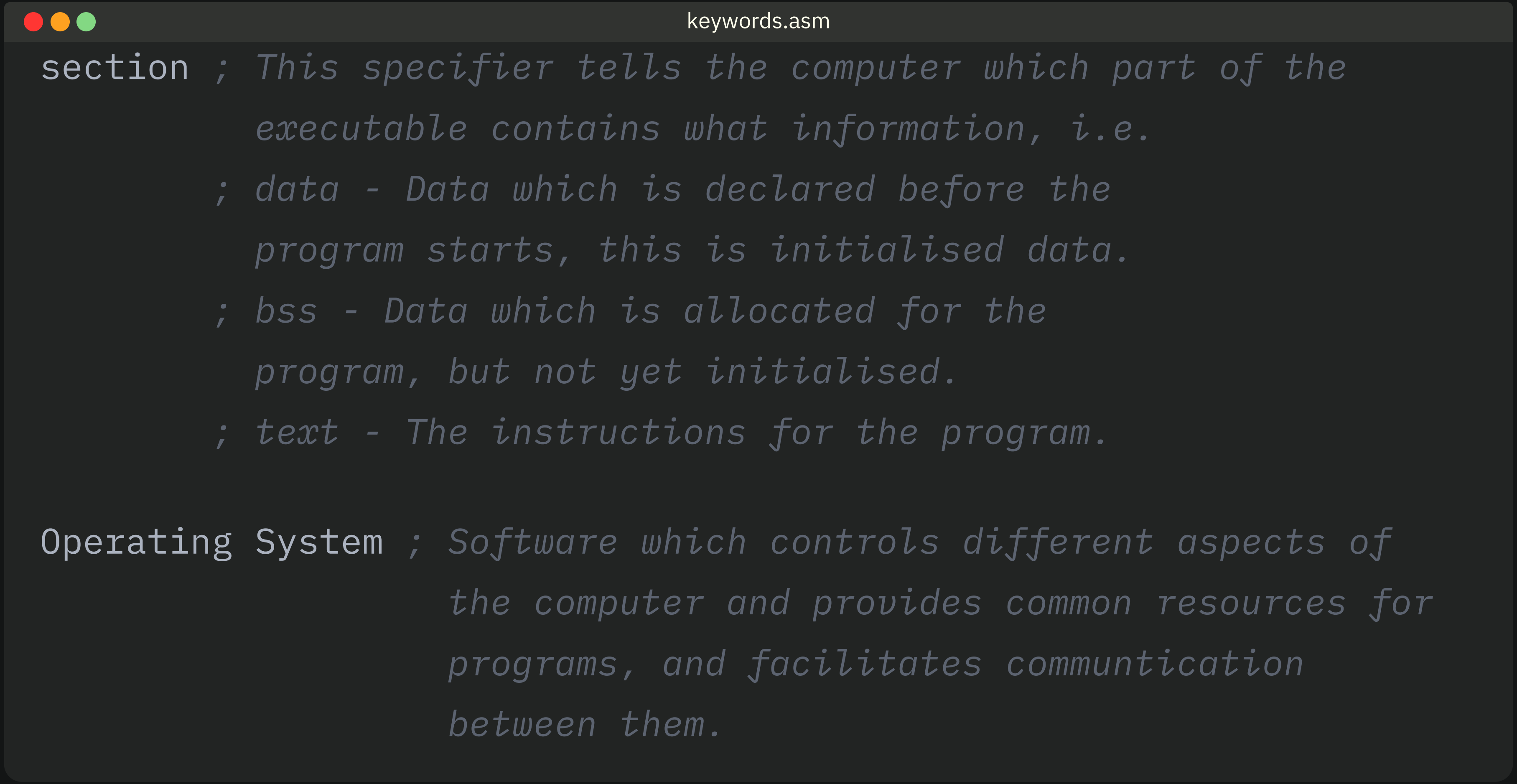
# Important Keywords in C



keywords.c

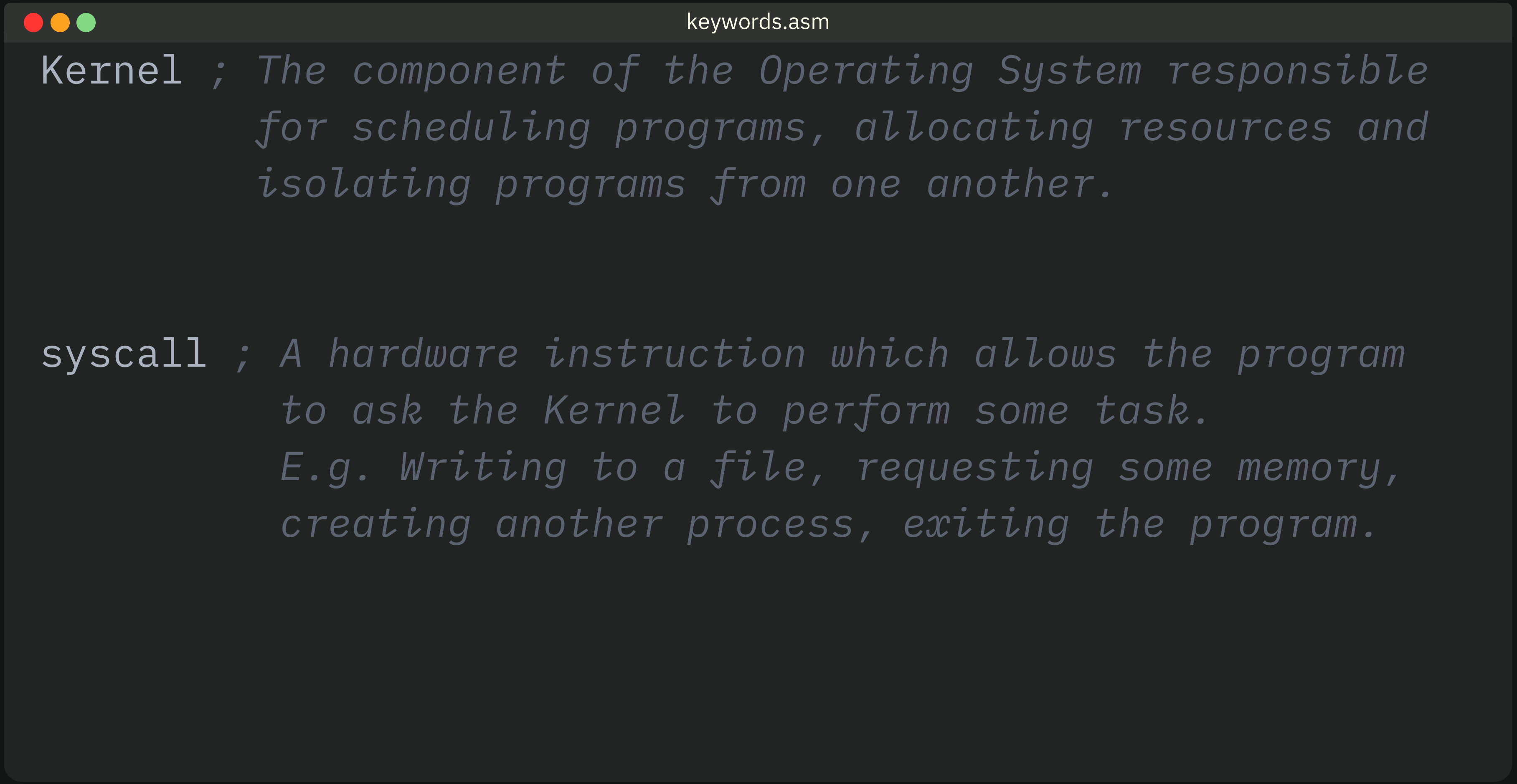
```
main()    // This function is defined as the entry
             point to code execution in C. You
             cannot write a C program without
             main()

int       // An 'int' is a data type in C which
             represents an signed Integer. On modern
             computers, it is 4 bytes large.

printf() // This functions prints to standard output.
             You can also give it format specifiers
             such as %d or %s to print different types
             of data
```

# Important Keywords in ASM


keywords.asm

```
section ; This specifier tells the computer which part of the
          executable contains what information, i.e.
        ; data - Data which is declared before the
          program starts, this is initialised data.
        ; bss - Data which is allocated for the
          program, but not yet initialised.
        ; text - The instructions for the program.

Operating System ; Software which controls different aspects of
                   the computer and provides common resources for
                   programs, and facilitates communtication
                   between them.
```

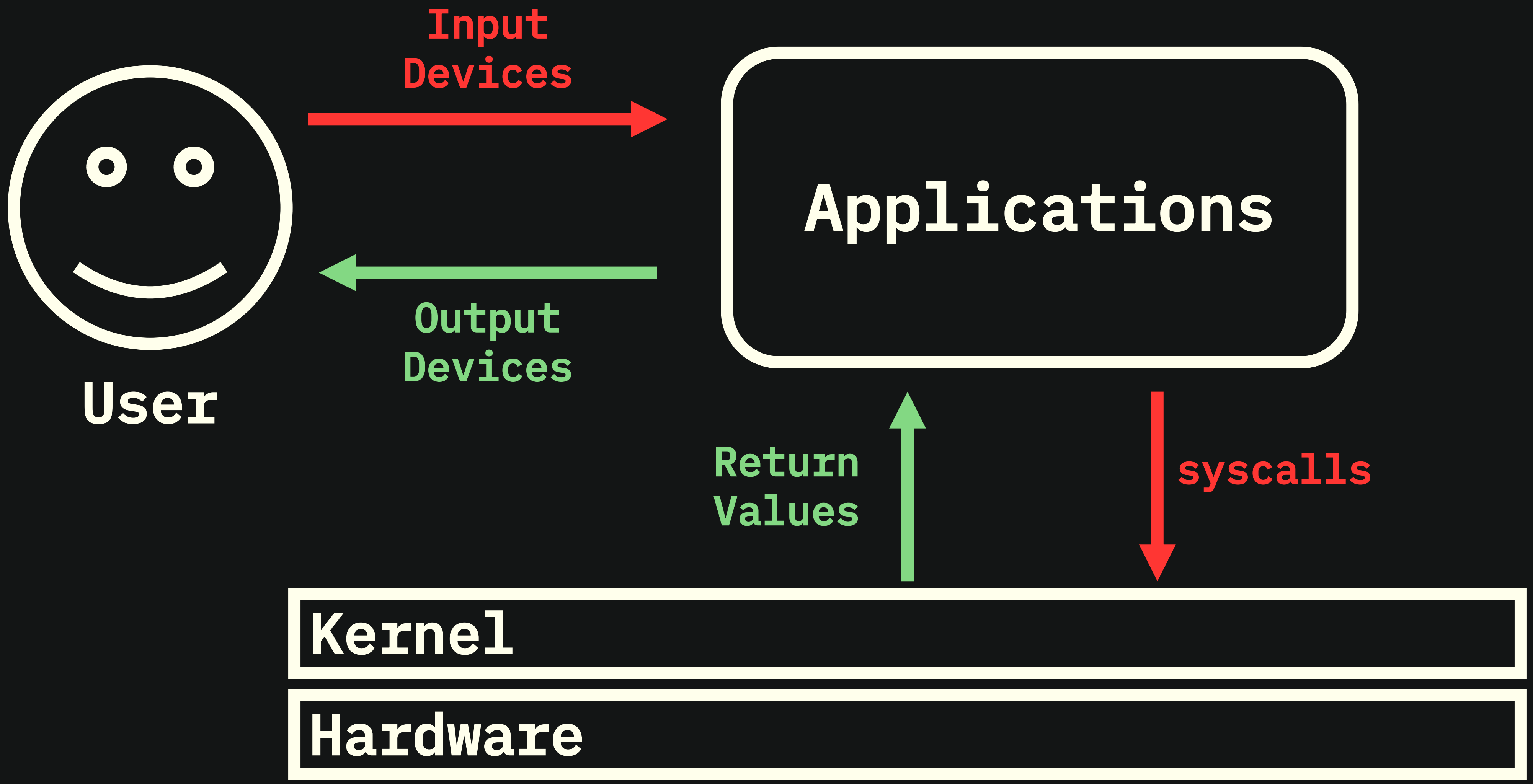
## More Important Keywords in ASM



keywords.asm

```
Kernel ; The component of the Operating System responsible
         for scheduling programs, allocating resources and
         isolating programs from one another.


syscall ; A hardware instruction which allows the program
          to ask the Kernel to perform some task.
          E.g. Writing to a file, requesting some memory,
          creating another process, exiting the program.
```

# Program Execution Diagram



## High Level CPython

And Python Bytecode.

## High Level CPython

We’ll take a look at the source code for CPython, build it from source and poke around to see what cool code is there for us!

## Observing Python’s Bytecode

We can use Godbolt to see the generated Bytecode which contains operations allow us to push and pop off the Stack as well as performing operations on the Stack.

We will look at implementing some of these operations when we make our own Stack-Based VM in Python.

# Observing Python's Bytecode

We car
conta
well

**!HOLD ON!**

What actually *is* a compiler?

We will look at implementing some of these operations when we make our own Stack-Based VM in Python.

## What are Compilers

And why CPython is so slow.

## What are Compilers

Godbolt shows us the generated bytecode, but we don’t know what compilers are yet.

A compiler is a tool which turns one language into another language. We have compilers that turn C -> Machine Code, Java -> JVM Bytecode or Python -> Python Bytecode.

All of these languages go from human-readable text to a form of Binary Instruction format.

We also can have compilers which convert between human-readable programming languages, Kotlin -> Java, CTML -> HTML, Svelte -> JS.

## Key Steps in Compilation

**Lexical Analysis:** Breaking the Source Code into its keywords then creating *tokens* from them.

**Syntax Analysis:** Ensuring all symbols are in the correct format, according to the syntax of the language. Then it builds an *abstract syntax tree (AST)*, verifies it is correct and reports errors if it finds any.

**Semantic Analysis:** Using the *AST*, the compiler tries to verify if the code is correct. This includes things like type-checking, function argument checks and correct variable usage.

## Key Steps in Compilation

**Intermediate Representation:** The *AST* is build into an intermediate representation like an *LLVM IR* or others.

**Optimisation:** The compiler will analyse the *IR* and manipulate it to be more by removing redundant function calls or utilising the special features of the hardware it is running on.

**Code Generation:** Once the compiler finished optimising the *IR*, it will start generating the code for the target device or interpreter.

## So why is CPython so *SLOW*

**No Optimisations:** Since CPython reads code line-by-line, it cannot perform any code manipulation to make it run faster.

**Heap Allocation:** Almost all of the data used in a Python program exists on the *heap*, a segment of memory with slower access speeds than other areas like the *Stack* or *Cache*.

**Cache Misses:** CPUs are very good at optimising what data should be in their cache at any time. Certain types of Python's data structures, like lists and dictionaries aren't stored in contiguous blocks in memory, making it difficult for Python programs to be cached by the CPU.

**Unknown Types:** Python doesn't do runtime type checking until operations are performed on them. This means the Python interpreter cannot optimise as many optimisation techniques require the knowledge of variable types at compile time.

## Other Python Implementations

And why we still use CPython.

## Other Python Implementations

**Pypy:** Uses both a Just-In-Time Compiler (JIT) to speed up code execution.

**Cython:** A compiler which first transpiles your Python code to C code, then uses your C compiler of choice to generate and run an executable.

**Jython:** A compiler which runs your Python code in the Java Virtual Machine (JVM), allowing you to use Java Libraries in your Python Code.

**IronPython:** An implementation of Python which allows you to use the .NET runtime.

# A Brief Look at Pypy and JITs.

Suppose we had the function:

```python
def add(a, b):
    return a + b
```

CPython doesn't know what a and b will be at runtime. It doesn't know if the data being passed to the function so it must re-check the types and perform the appropriate operation every time the function is called.

## A Brief Look at Pypy and JITs.

Since Pypy is Just-In-Time (JIT), it dynamically analyses our code at runtime.

Pypy profiles the code and optimises functions by seeing what types of data it normally takes.

Using the add function, Pypy will attempt to spot a pattern of what is passed to it. If it sees only integers are passed, it will recompile the function to use only integer additions.

If it is passed variables that aren’t ints, it will return to the original slower bytecode for the function.

## Why do we still use CPython then?

CPython is the main implementation of the language, it is always the most up-to-date to the Python specs.

It can easily communicate with C code, allowing libraries like *numpy* and others to interop with it.

It also has a strong ecosystem of tools for using, creating and analysing making it the most usable in the industry.

## Today’s Exercise

## Today’s Exercise

Creating our own Stack-Based Virtual Machine!

This allows us to see the steps in making a programming language.

## The End

Thanks for coming!

Notes uploaded on the Github Repository.

See you next week!