Python

What are we going to cover ?

language,

Fundamentals

Introduction

Pycharm

Setup

variables

Data Types

Operators

control, loop, ... spacing

Statements

indentation

empty, named, unnamed, (lambda)

Functions

map, filter, reduce, flask .....

Functional Programming

class, reused, abstraction, encapsulation.

OOP

Modules and Packages

Advanced Features

REST – Flask

Web Services

numeric – arrays - collections

Numpy

Excel on steroid

Pandas

Data Analysis

charts / graphs / tables – matplotlib

Data Visualization

selenium – data scraping

Testing

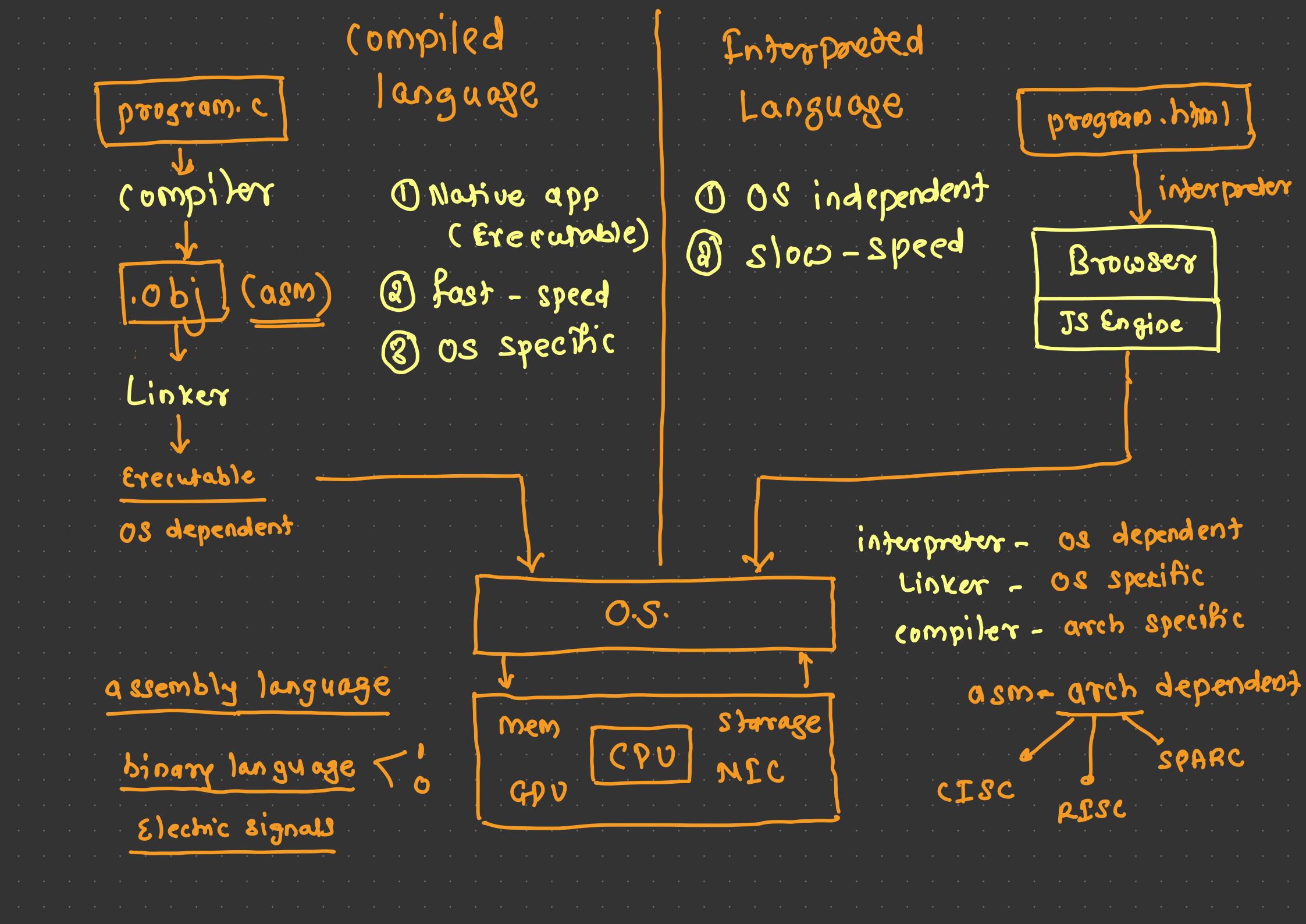
Scrapy.

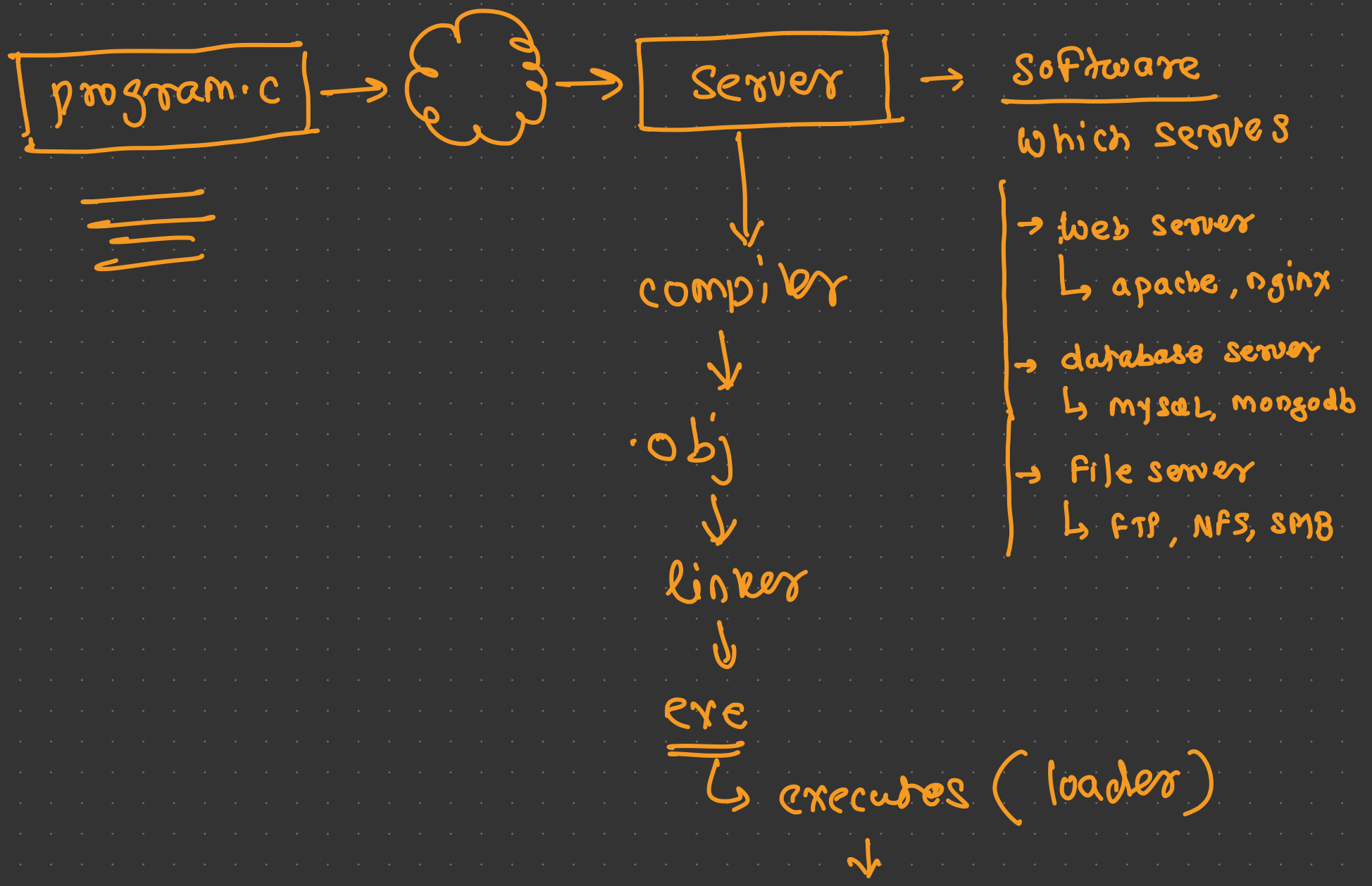
Web Scrapping

About Instructor

- 16+ years of experience
- Associate Technical Director at Sunbeam
- Freelance Developer
- Worked in various domains using different technologies
- Developed 180+ mobile applications on iOS and Android platforms
- Developed various websites using PHP, MEAN and MERN stacks
- Languages I love: C, C++, Python, JavaScript, TypeScript, PHP, go

Language Fundamentals


compiled language
program.c
compiler
.obj (asm)
Linker
Executable
OS dependent
① Native app (Executable)
② fast - speed
③ OS specific
Interpreted Language
① OS independent
② slow - speed
program.html
interpreter
Browser
JS Engine
O.S.
mem
CPU
storage
GPU
NIC
assembly language
binary language
Electric signals
interpreter - OS dependent
Linker - OS specific
compiler - arch specific
asm - arch dependent
CISC
RISC
SPARC


program.c
Server
Software
Which serves
web server
apache, nginx
database server
mysql, mongodb
file server
FTP, NFS, SMB
compiler
.obj
linker
exe
executes (loader)

What is a computer language?

- A language is a medium to interact with computer
- A language is used to solve a problem by giving some solution (writing a program)
- Types
  - **Machine languages**: the code written in 0s and 1s and can be directly executed by CPU
  - **Assembly languages**: a language closely related to one or a family of machine languages, and which uses mnemonics to ease writing
  - Programming languages
    - **General purpose languages**: a programming language that is broadly applicable across application domains, and lacks specialized features for a particular domain
    - **Markup languages**: a grammar for annotating a document in a way that is syntactically distinguishable from the text
    - **Stylesheet language**: a computer language that expresses the presentation of structured documents
    - **Configuration language**: a language used to write configuration files
    - **Query language**: a language used to make queries in databases and information systems
    - **Scripting languages**: a language used to write simple to complex scripts

Low Level vs High Level Languages

High Level Language	Low Level Language
It is programmer friendly language	It is a machine friendly language
High level language is less memory efficient	Low level language is high memory efficient
It is easy to understand	It is tough to understand
It is simple to debug	It is complex to debug comparatively
It is simple to maintain	It is complex to maintain comparatively
It is portable	It is non-portable
It can run on any platform	It is machine-dependent
It needs compiler or interpreter for translation	It needs assembler for translation
E.g. Python	E.g. Assembly

Compiled language → Native app

- A programming language which involves an executable to execute the logic instead of executing the source code directly
- E.g. C, C++, Pascal, Objective-C, Swift etc.
- Stages
  - Pre-processing
    - Used to preprocess the code
    - Comments removal
    - Code expansion
    - Conditional compilation
  - Compiling
    - Compiler is used to translate pre-processed code into assembly instructions specific to the target processor architecture
  - Assembling
    - Assembler is used to translate the assembly instructions to object code
  - Linking
    - Linker used in this stage re-arranges the code and insert missing files to emit an executable

Interpreted Language

- A language mostly executes source code directly and freely, without previously compiling a programm into an executable
- E.g. Python, JavaScript, Perl, BASIC etc.
- Interpretation
  - Interpreter used in the language executes the high level source code directly

Compiler vs Interpreter

Compiler	Interpreter
The compiler is faster, as conversion occurs before the program executes.	The interpreter runs slower as the execution occurs simultaneously.
Errors are detected during the compilation phase and displayed before the execution of a program.	Errors are identified and reported during the given actual runtime.
Compiled code needs to be recompiled to run on different machines.	Interpreted code is more portable as it can run on any machine with the appropriate interpreter.
It requires more memory to translate the whole source code at once.	It requires less memory than compiled ones.
Debugging is more complex due to batch processing of the code.	Debugging is easier due to the line-by-line execution of a code.

Python

Introduction

- Python is a general purpose, high level and interpreted language
- It is dynamically typed and garbage collected language
- It supports programming paradigms like
  - Procedural programming
  - Object oriented programming
  - Functional programming
  - Aspect oriented programming (metaprogramming and magic metaobjects)
- Heavily dependent on indentation to create blocks which makes it very easy to read the code
- It has more than 300,000 packages included with wide range of functionality
  - Graphical user interfaces
  - Web frameworks
  - Networking
  - Automation
  - Web scraping

History

- Python was conceived in the late 1980s by Guido Rossum in Netherlands
- It is considered as a successor to the ABC language (itself inspired by SETL)
- Its implementation began in December 1989
- Van Rossum continued as Python's lead developer until July 12, 2018
- When he announced his permanent vacation from his responsibilities, a team of five members was developed in Jan 2019 to lead the project

Advantages of Python

- Python is simple to use, but it is a real programming language, offering much more structure and support for large programs than shell scripts or batch files can offer
- Python also offers much more error checking than C, and, being a very-high-level language, it has high-level data types built in, such as flexible arrays and dictionaries
- Because of its more general data types Python is applicable to a much larger problem domain than Awk or even Perl, yet many things are at least as easy in Python as in those languages
- Python allows you to split your program into modules that can be reused in other Python programs
- Python is an interpreted language, which can save you considerable time during program development because no compilation and linking is necessary
- Python enables programs to be written compactly and readably
- Programs written in Python are typically much shorter than equivalent C, C++, or Java programs, for several reasons:
  - the high-level data types allow you to express complex operations in a single statement
  - statement grouping is done by indentation instead of beginning and ending brackets
  - no variable or argument declarations are necessary

Differences between Python and C Language

C Language	Python
C is procedure-oriented programming language. It does not contain the features like classes, objects, inheritance, polymorphism, etc.	Python is object-oriented oriented language. It contains features like classes, objects, inheritance, polymorphism, etc.
Pointers concept is available in C	Python does not use pointers
C has switch statement	Python does not have switch statement
C programs execute faster	Python programs are slower compared to C. PyPy flavor or Python programs run a bit faster but still slower than C
Compulsory to declare the datatypes of variables, arrays etc	Type declaration is not required in Python
C does not have exception handling facility and hence C programs are weak	Python handles exceptions and hence Python programs are robust
C does not contain a garbage collector	Automatic garbage collector is available in Python
C supports in-line assignments	Python does not support in-line assignments
Indentation of statements is not necessary in C	Indentation is required to represent a block of statements

Versions

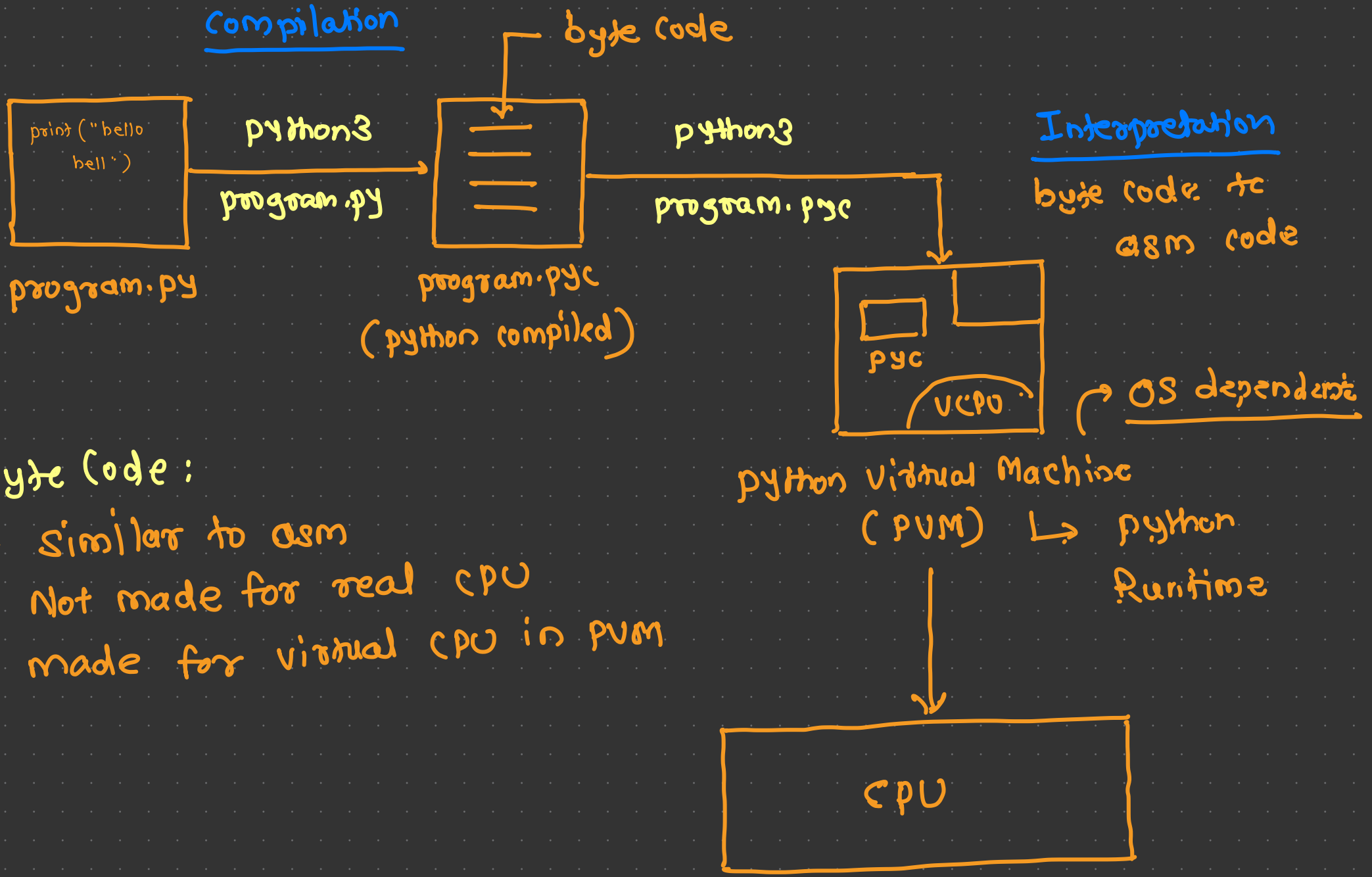
- Version 0.9.0 [Feb 1991]
  - Having features like classes with inheritance, exception handling, functions etc.
  - One of the major versions of python
- Version 1 [Jan 1994]
  - The major new features included in this release were the functional programming tools like labda, map, filter, reduce
  - The last version released was 1.6 in 2000
- Version 2 [Oct 2000]
  - Introduced features like list comprehension, garbage collection, generators etc.
  - Introduced its own license known as Python Software Foundation License (PSF)
  - The last version released was 2.7.16 in Mar 2019
- Version 3 [Dec 2008]
  - Python 3.0 is also called "Python 3000" or "Py3K"
  - It was designed to rectify fundamental design flaws in the language
  - Python 3.0 had an emphasis on removing duplicative constructs and modules
  - The last version released was 3.7.4 in Jul 2019

Environment Setup

- Ubuntu
  - Install python3 on using apt installer
    - > sudo apt-get install python3 python3-pip
  - Download pycharm community edition
    - https://www.jetbrains.com/pycharm/download

What have you downloaded

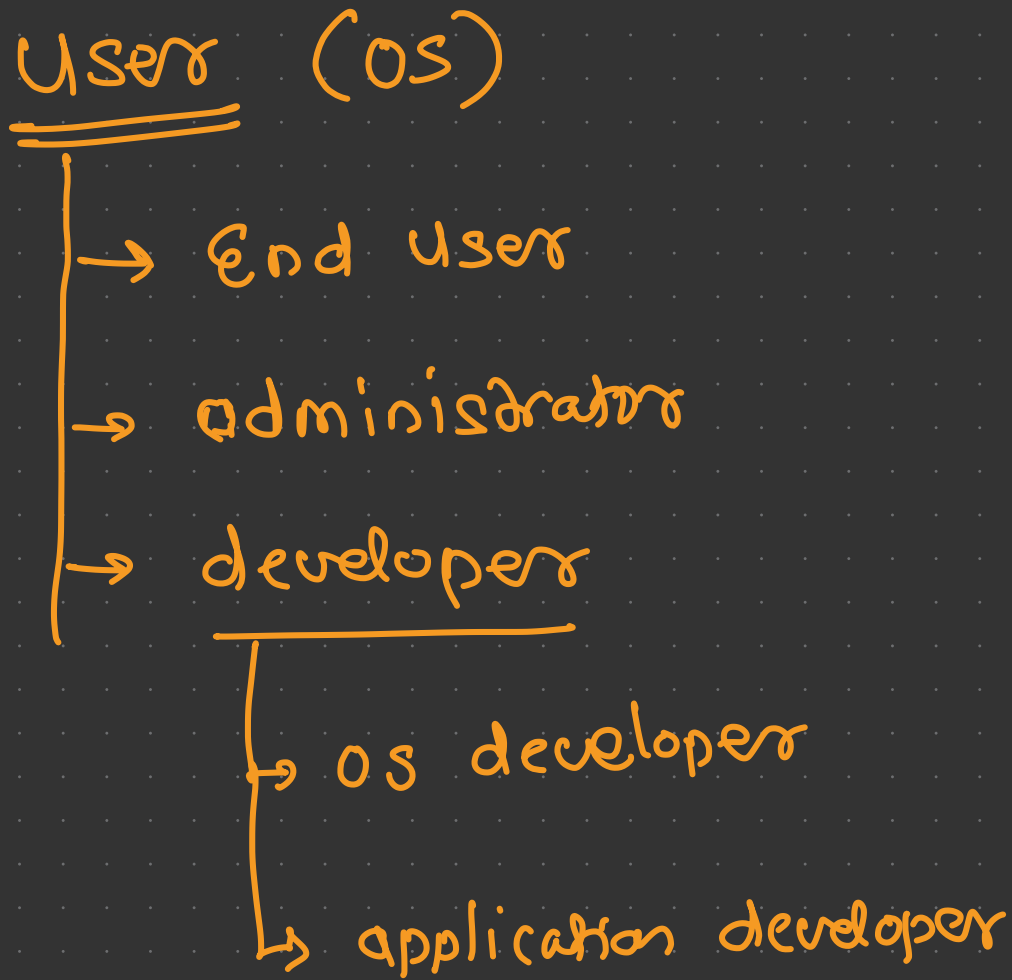
- Python has many versions like
  - CPython → python interpreter is developed in "C"
  - Pypy → " " in python
  - Jython → " " in Java
  - InronPython → " " in Iron
- The standard version is called as CPython which is what most of us get when we download from the official site
- Henceforth when we say we are using python, it will be CPython



Byte code:

- similar to asm
- Not made for real CPU
- made for virtual CPU in PVM

	OS	CPU arch
compiler	independent	dependent
linker	dependent	dependent
interpreter	dependent	independent
.obj	independent	dependent
.exe	dependent	dependent
.py	independent	independent
.pyc	〃	〃


User (OS)
End user
administrator
developer
OS developer
application developer

Hello world program explained

- Python is not a compiled language: misconception
- CPython has a compile base
- Python compiles into bytecodes
- CPU can not understand bytecodes
- We need an interpreter to understand and execute the bytecodes
- This interpreter is nothing but the python virtual machine

Python Virtual Machine (PVM)

- Python compiler compiles the python source code into byte codes
- Byte codes represent fixed set of instructions to perform all types of operations
- The size of each byte code is 1 byte or 8 bits and hence the these are called as byte codes
- The role of PVM is to convert these byte codes into machine code so that the computer can execute those machine code instructions and display the final output
- To carry out this conversion, PVM is equipped with an interpreter
- The interpreter converts the byte code into machine code and sends that machine code to the computer processor for execution
- Since interpreter is playing the main role, often the Python Virtual Machine is also called an interpreter

Foundation

Identifier and keywords

num = 200 if = 200 X

- Identifiers
  - Identifiers or symbols are the names you supply for variables, types, functions, and classes
  - Identifier names must differ in spelling and case from any keywords
- Keywords
  - Keywords are the identifiers which are reserved
  - They can not be used to naming variables, types, functions or classes
  - There are 35 keywords in python 3.8
    - Value Keywords: True, False, None
    - Operator Keywords: and, or, not, in, is
    - Control Flow Keywords: if, elif, else
    - Iteration Keywords: for, while, break, continue, else
    - Structure Keywords: def, class, with, as, pass, lambda
    - Returning Keywords: return, yield
    - Import Keywords: import, from, as
    - Exception-Handling Keywords: try, except, raise, finally, else, assert
    - Asynchronous Programming Keywords: async, await
    - Variable Handling Keywords: del, global, nonlocal

num = None && ||

Rules and Conventions

- Constant and variable names should have a combination of letters in lowercase (**a to z**) or uppercase (**A to Z**) or digits (**0 to 9**) or an underscore (_)
  - If you want to create a variable name having two words, use underscore to separate them
- Create a name that makes sense (use meaningful names)
- Use capital letters possible to declare a constant ✗ NUM = 200 (constant)
- Never use special symbols like !, @, #, $, %, etc.
- Don't start a variable name with a digit
- Identifiers are case sensitive

first name = "steve" ✗ (space ✗)

1name = "steve" ✗ (digit)

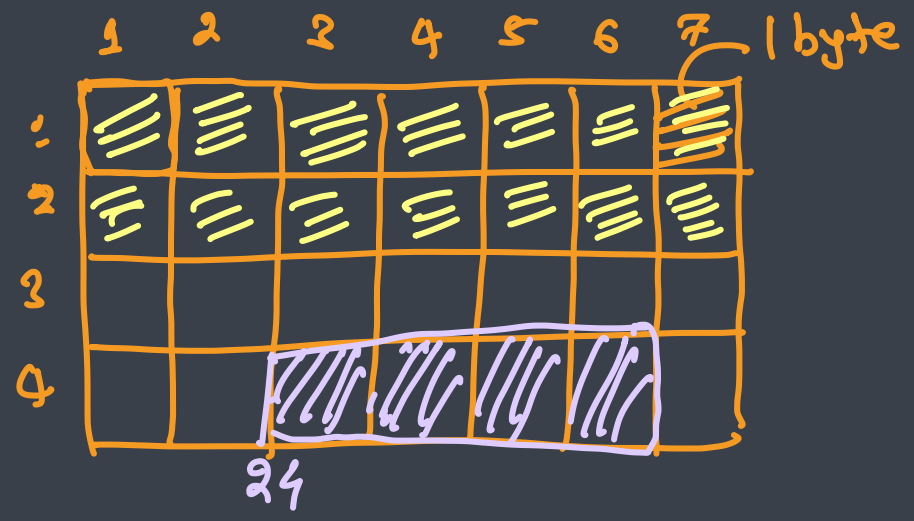
num Num

first.name = "steve" ✗ (dot ✗)

first_name = "steve" ✓

Variable

- A variable is a named location used to store data in the memory
- It is helpful to think of variables as a container that holds data that can be changed later
- E.g.
  - value = 20
  - name = "steve"



num = 2 → 00000010

4 bytes

num = 0x24

num = 20

Symbol	data type	Value	address
num	int	2	0x24

Symbol table

(compilation)

Constants – convention

- Constants are written in all capital letters and underscores separating the words
- E.g.
  - PI = 3.14

- In reality, we don't use constants in Python
- Naming them in all capital letters is a convention to separate them from variables, however, it does not actually prevent reassignment

Statements

- Instructions that a Python interpreter can execute are called statements
  - E.g., a = 1 is an assignment statement
- **Single line statements**
  - The end of a statement is marked by a newline character
- **Multi-line statement**
  - We can make a statement extend over multiple lines with the line continuation character (\)
  - Line continuation is implied inside parentheses ( ), brackets [ ], and braces { }
  - We can also put multiple statements in a single line using semicolons
- **Comments**
  - Comments are very important while writing a program
  - They describe what is going on inside a program, so that a person looking at the source code does not have a hard time figuring it out
  - In Python, we use the hash (#) symbol to start writing a comment

num = 20

name = "abc"

int num; → variable

data type

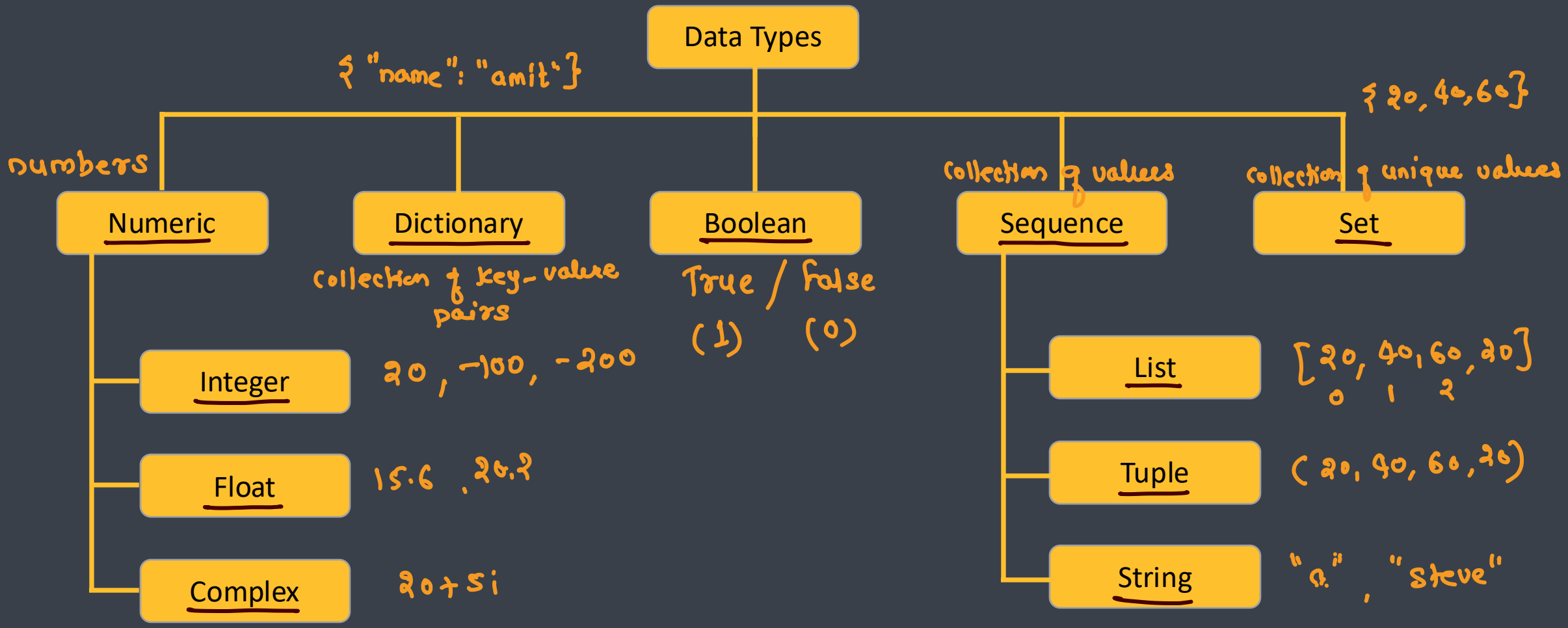
Data Types

1) You CANNOT get size of data type
2) You CAN NOT get address of variable

Need of Data Types

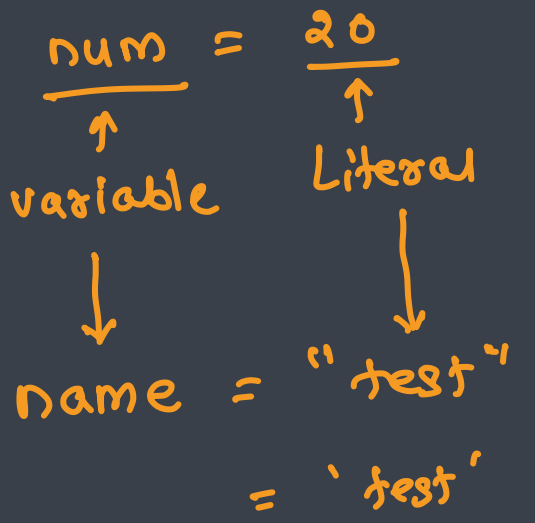
- Data type is an attribute associated with a piece of data that tells a computer system how to interpret its value
- Understanding data types ensures that data is collected in the preferred format and the value of each property is as expected
- It helps to know what kind of operations are often performed on a variable

Python Data Types



Literals

- Literal is a raw data given in a variable or constant



- **Numeric Literals**
  - Numeric Literals are immutable (unchangeable)
  - Numeric literals can belong to 3 different numerical types:
    - Integer (value = 100)
    - Float (salary = 15.50)
    - Complex (x = 10 + 5j)

- **String literals**
  - A string literal is a sequence of characters surrounded by quotes
  - We can use both single, double, or triple quotes for a string
  - E.g., name = “steve” or name = ‘steve’

single / double / triple

Literals

- **Boolean literals**
  - A Boolean literal can have any of the two values: True or False
  - E.g., can_vote = True or can_vote = False

- **Special literal**
  - Python contains one special literal i.e. None
  - We use it to specify that the field has not been created
  - E.g., value = None

- **Literal Collections**
  - Used to declare variables of collection types
  - There are four different literal collections List literals, Tuple literals, Dict literals, and Set literals

Static typing

- compiler checks the type of variable
- compiler assigns datatype at compilation
- dev must provide data type to every var.
- c, c++, java

Dynamic typing

- data type will be assigned at the time of running app
- data type will be inferred (automatically assigned by language)
- python, js

Strictly typed
Strongly typed

int num;

num = 20 ; ✓

num = "test" ; ✗

- C, C++, java, swift

weakly typed

num = 20 (int)

num = "test" (str)

- python, js

Type Hinting

- Due to the dynamic nature of Python, inferring or checking the type of an object being used is especially hard
- This fact makes it hard for developers to understand what exactly is going on in code they haven't written. As a result they resort to trying to infer the type with around 50% success rate.
- **Why type hints?**
  - **Helps type checkers:** By hinting at what type you want the object to be the type checker can easily detect if, for instance, you're passing an object with a type that isn't expected
  - **Helps with documentation:** A third person viewing your code will know what is expected where, ergo, how to use it without getting them TypeErrors
  - **Helps IDEs develop more accurate and robust tools:** Development Environments will be better suited at suggesting appropriate methods when know what type your object is

Type Conversion → type casting

- The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion
- Python has two types of type conversion.
  - Implicit Type Conversion
    - Python automatically converts one data type to another data type
    - This process doesn't need any user involvement
    - E.g. int float
      - result = 10 + 35.50
  - Explicit Type Conversion - manual
    - Users convert the data type of an object to required data type
    - We use the predefined functions like int(), float(), str(), etc to perform explicit type conversion
    - E.g.
      - num_str = str(1024)

↳ if possible

↳ only smaller type gets converted to bigger one

Type Conversion

- Type Conversion is the conversion of object from one data type to another data type
- Implicit Type Conversion is automatically performed by the Python interpreter
- Python avoids the loss of data in Implicit Type Conversion
- Explicit Type Conversion is also called Type Casting, the data types of objects are converted using predefined functions by the user
- In Type Casting, loss of data may occur as we enforce the object to a specific data type

Operators

Operators

- Operators are special symbols in Python that carry out arithmetic or logical computation
- The value that the operator operates on is called the operand
- Types
  - Arithmetic operators
  - Comparison operators
  - Logical operators
  - Bitwise operators
  - Special operators
  - Membership operators

Arithmetic Operators

Operator	Meaning	Example
+	Add two operands or unary plus	x + y + 2
-	Subtract right operand from the left or unary minus	x – y
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y

Arithmetic Operators

- Precedence
  - Decides how the expression will be solved
  - Brackets ()
  - Power Of (**)
  - Multiplication (*)
  - True Division (/)
  - Floor Division (//)
  - Addition (+)
  - Subtraction (-)
- Associativity
  - When there are two ore more operators in an expression with same precedence then associativity is used to decide the way to solve the expression
  - All the operators have left to right associativity except power operator (which is right to left)

Comparison operators → returns boolean

Operator	Meaning	Example
>	Greater than - True if left operand is greater than the right	x > y
<	Less than - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

Logical operators → work on boolean operands & return boolean answer

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

and operator

T	T	T
T	F	F
F	T	F
F	F	F

or operator

T	T	T
T	F	T
F	T	T
F	F	F

not operator

T	F
F	T

Assignment Operators

Operator	Meaning	Example
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5

Operator	Meaning	Example
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

Bitwise operators

Operator	Meaning	Example
&	Bitwise AND	x & y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x >> 2 = 2 (0000 0010)
<<	Bitwise left shift	x << 2 = 40 (0010 1000)

Special operators

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Membership Operators

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x