Python

What are we going to cover ?

beginner

language, types

Fundamentals

history, advantages, ...

Introduction

environment → windows / linux / mac os

Setup

variable / constant || inferred || 5 types

Data Types

maths, comparison, logical, bitwise

Operators

code || control statement / loop statement / comment

Statements

print ("hello")

intermediate

reuseble unit

Functions

function is a first class citizen || map / filter / reduce

Functional Programming **

classes / objects / inheritance / .....

OOP

reusability → 3000 packages

Modules and Packages

Generators / closure / decorator / iterator

Advanced Features

REST services → flask

Web Services

advanced level

array → faster

Numpy

Data frame

Pandas

understand

Data Analysis

matplotlib / seaborn / plotly → graphs / diagrams

Data Visualization

selenium → data scraping

Testing

Beautiful Soup, Selenium

Web Scrapping

About Instructor

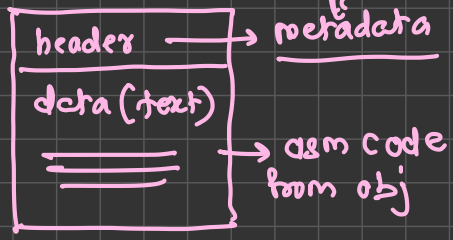
- More than 18+ years of experience
- Associate Technical Director at Sunbeam and a Freelance Developer working with real world projects with overseas clients
- Developed numerous mobile applications on iOS and Android platforms
- Developed various websites using LAMP, MEAN and MERN stacks
- Languages I love: C, C++, Python, JavaScript, TypeScript, PHP, Go
- Pursuing PhD in Computer Application (Machine Learning)

Swift java/ kotlin hybrid → React native

Language Fundamentals

Compilation

- magic number
- date time stamp
- file format
- data section



Executable file

program.c

↓

compiler → architecture dependent

↓

object file (obj) → assembly code

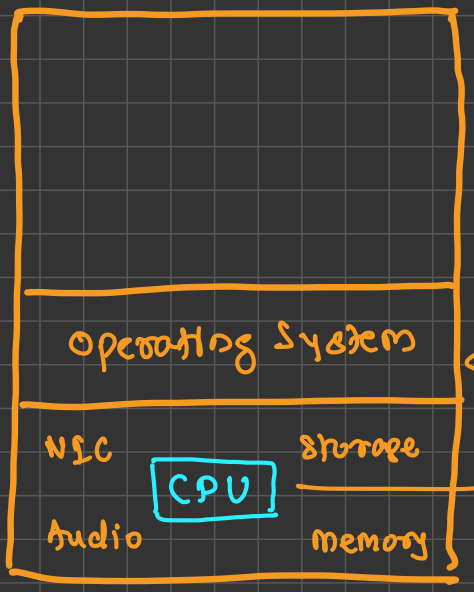
architecture dependent

↓

Linker → OS dependent (links the obj file with all definitions / libraries) +

↓

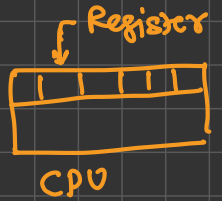
Executable file → OS dependent add file header required by OS executable file format



Machine

CPU is mechanical device

- electric current (+5V, +2V)
- assembly language → opcodes → assembler → opcodes → OS / IS → current
  - ADD Ax Bx
  - architecture specific



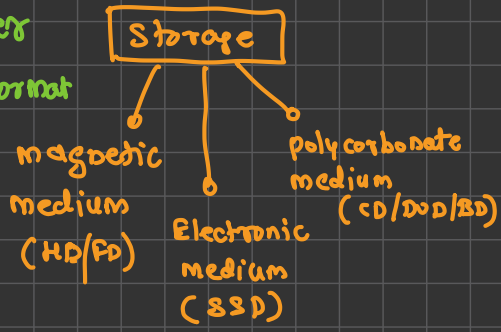
CPU architectures:

desktop → CISC → Complex Instructions Set computing
x86 / x64

mobile → RISC → Reduced ISC → ARM

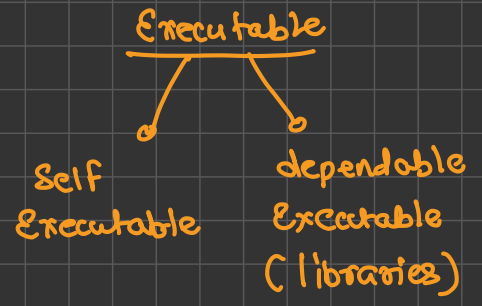
ARM - Advanced RISC Machine

→ SPARC → Sun Processor Architecture



Executable file format

	Format	Self Executable	Libraries	
			static	dynamic
Windows	PE COFF → Portable Executable Common Object File Format	.exe	.lib library	.dll dynamic linking lib
Linux	ELF → Executable and Linkable File format	.o, .out	.ar archived	.so shared object
macOS	MACH-O	.o, .out	.ar archived	.dylib dynamic library



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

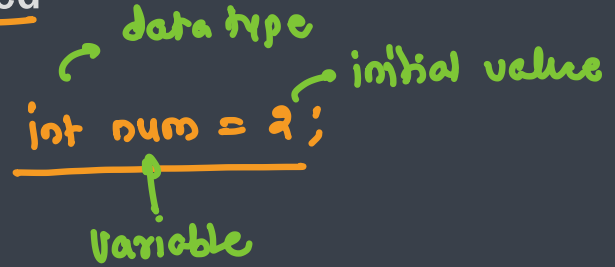
print("hello")

High Level Language (python)	Low Level Language → asm
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 → CPU arch dependent
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

compiled language → compilation → Executable file → Execute
compiler platform dependent

- Compiled languages are a category of programming languages where the source code is translated into machine language or another intermediate form before it can be executed by a computer
- This translation process is typically handled by a compiler, which converts the high-level code into executable instructions that can run on a specific hardware platform → CPU architecture
- Compiled languages generally offer better performance than interpreted languages but are less portable because the compiled code is specific to a particular architecture
- E.g. C, C++, Go, C#
- Key Characteristics
  - Faster execution speed
  - Memory efficient
  - Static Typing →



Interpreted Language

interpreted language program → interpretation / interpreter → Execute the code

- Interpreted Languages are a category of programming languages for which most of their implementations execute instructions directly and freely, without previously compiling a program into machine-language instructions → No executable file created
- The interpreter executes the program directly, translating each statement into a sequence of one or more subroutines and immediately carrying out the actions
- This is in contrast to compiled languages, where the code is translated into machine language before execution
- E.g. Python*, JavaScript, Perl, Ruby
- Key characteristics
  - Slower execution speed compared to compiled languages
  - More portable than compiled languages
  - Easy to debug
  - Dynamic Typing → data type will be inferred → num = 2 (variable, initial)
  - Memory management is simpler than compiled languages

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

- console
- web app
- scripts
- AI/ML apps

- procedure oriented programming
- objected oriented programming
- functional programming
- aspect oriented programming

- **Programming Language**: Python is a high-level, general-purpose programming language
- **Created By**: Developed by Guido van Rossum and first released in 1991
- **Readability**: Known for its simple and clean syntax, emphasizing code readability
- **Interpreted**: Python is an interpreted language, meaning code is executed line by line *
- **Multi-Paradigm**: Supports multiple programming styles, including:
  - Procedural
  - Object-Oriented (OOP)
  - Functional
- **Cross-Platform**: Runs on various operating systems like Windows, macOS, and Linux → write once and run anywhere
- **Open Source**: Freely available and supported by a large, active community
- **Extensive Libraries**: Comes with a rich standard library and third-party packages for diverse applications > 300k packages
- **Versatile**: Used in web development, data science, machine learning, automation, scientific computing, and more
- **Dynamically Typed**: Variables do not require explicit type declaration → data type inferred

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

- python now is community driven project

Key Features

- Easy to Learn and Use
  - Simple and readable syntax, making it beginner-friendly
  - Emphasizes code readability with indentation and minimalistic design
- Interpreted Language
  - Code is executed line by line, without the need for compilation
  - Enables rapid development and testing
- Cross-Platform Compatibility
  - Runs on multiple operating systems, including Windows, macOS, and Linux
- Dynamically Typed
  - No need to declare variable types explicitly; types are inferred at runtime
- Extensive Standard Library
  - Comes with a rich set of built-in modules for tasks like file I/O, regular expressions, and web development
- Supports Multiple Programming Paradigms
  - Procedural, object-oriented, and functional programming styles

Key Features

- Open Source and Free
  - Freely available for use and modification, with a large community contributing to its development
- Large Ecosystem of Libraries and Frameworks
  - Libraries for data science (Pandas, NumPy), machine learning (TensorFlow, PyTorch), web development (Django, Flask), and more
- High-Level Language
  - Abstracts low-level details, allowing developers to focus on problem-solving
- Automatic Memory Management → garbage collection
  - Features garbage collection to manage memory allocation and deallocation
- Community Support
  - Active and growing community, providing extensive documentation, tutorials, and third-party tools
- Scalable and Versatile
  - Suitable for small scripts as well as large-scale applications
- Integration Capabilities
  - Easily integrates with other languages like C, C++, and Java

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 VS Code for Linux
    - https://code.visualstudio.com/download
    - Install python extension for code IntelliSense
- Windows
  - Download the python from
    - https://www.python.org/downloads/
  - Download VS Code for Windows
    - https://code.visualstudio.com/download
    - Install python extension for code IntelliSense

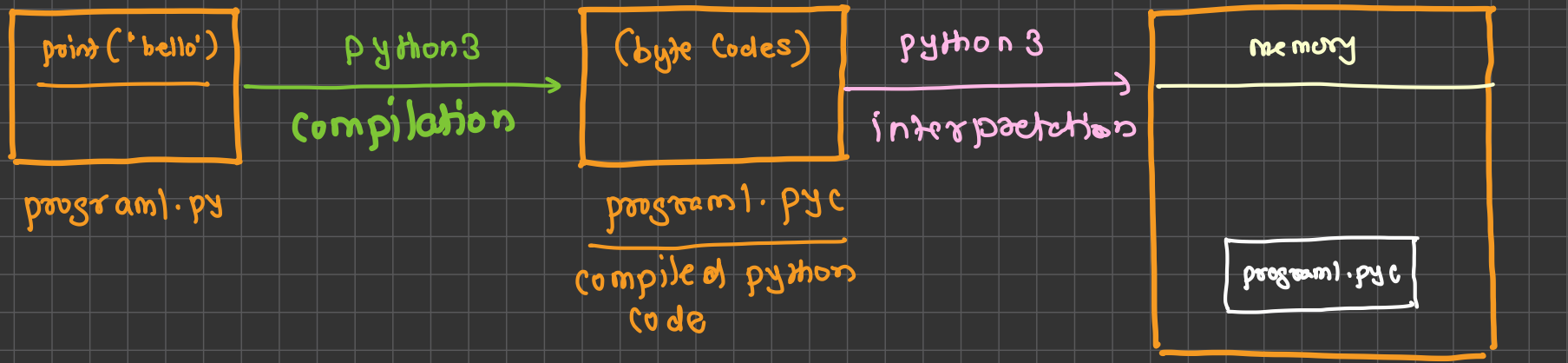
What have you downloaded

- Python has many versions like (compiler & interpreters)
  - CPython → written in C
  - Pypy → written in python
  - Jython → written in Java
  - IronPython → written in .Net
- 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

CPython

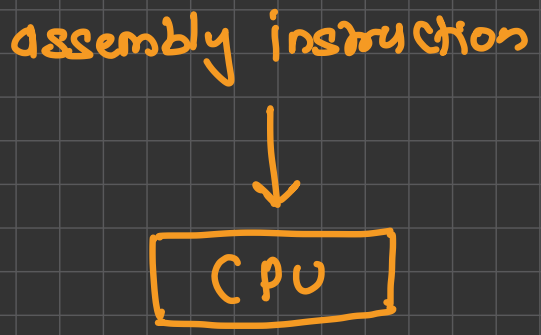
compiler
interpreter
→ python3

- CPython is the reference implementation of the Python programming language
- It is written in C and serves as the standard interpreter for executing Python code
- CPython is an essential component of the broader Python ecosystem and provides the core functionality that developers interact with when working with Python
- Key Features
  - Reference Implementation: CPython is the official interpreter for Python and serves as a reference implementation against which other implementations (like Jython, IronPython, and PyPy) are compared.
  - Written in C: The core components of CPython, including the Python interpreter and built-in modules, are written in C. This allows for efficient implementation and integration with system-level operations.
  - Interpreted Language: Despite being written in C, Python is an interpreted language. CPython interprets the high-level Python source code at runtime and converts it into bytecode, which is then executed on the virtual machine.
  - C API: CPython provides a C Application Programming Interface (API), enabling developers to write Python extensions in C. This is useful for optimizing performance-critical parts of applications. → numpy



* byte codes → intermediate asm code
  developed for Python Virtual Machine (PVM)
  → Not compatible with the CPU
  → hence, 'pyc cannot be executed by CPU directly

* by making PVM as dependent, the python code can be as independent



CPython

- **Bytecode Compilation**
  - **Bytecode Generation:**
    - When a Python script is executed, CPython first compiles the source code into bytecode. Bytecode is a lower-level, platform-independent representation of the source code → PVM is platform dependent
  - **Execution by Virtual Machine:**
    - CPython's virtual machine (PVM - Python Virtual Machine) executes the bytecode. This abstraction allows for portability and flexibility, as the bytecode can be run on any platform that supports Cpython

- **Standard Library and Built-in Modules**
  - **Extensive Standard Library:**
    - CPython comes with a comprehensive standard library that provides access to a wide range of functionality, including file I/O, system calls, internet protocols, and more.
  - **Built-in Modules:**
    - CPython includes a collection of built-in modules written in C, which enhance the language's capabilities and performance.

Hello world program explained

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

PVM → interpretation (byte code → asm code)

- PVM is the runtime engine of Python that executes the bytecode compiled from Python source code
- After Python code is compiled into bytecode (.pyc files), the PVM interprets and executes this bytecode.
- The PVM is the final layer of the Python interpreter, which includes:
  - Compiler: Converts source code to bytecode
  - PVM: Executes the bytecode
- Platform Independence:
  - The PVM ensures Python code can run on any platform where Python is installed, making Python a cross-platform language
- Not a Separate Program:
  - The PVM is not a standalone program but is embedded within the Python interpreter

PVM

- **Execution Process:**
  - Python source code (.py files) is compiled into bytecode
  - The bytecode is executed by the PVM
  - The PVM translates bytecode into machine-specific instructions for the CPU
- **Performance:**
  - The PVM is slower than directly executing machine code (e.g., C/C++ programs) because it interprets bytecode at runtime
  - However, tools like Just-In-Time (JIT) compilers (e.g., PyPy) can optimize PVM performance
- **Garbage Collection:** automatic memory management
  - The PVM also handles memory management, including garbage collection, to free up unused memory

Foundation

Identifier and keywords

if = 200 ✗ num = 200 def function(): pass class Person: pass

- 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 by language
  - 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

Rules and Conventions

No special symbols are allowed

- 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

conventions

- Never use special symbols like !, @, #, $, %, etc.
- Don't start a variable name with a digit
- Identifiers are case sensitive

num = 200 → variable

NUM = 200 → constant

first name = "amit" ✗

first_name = "amit" ✓

first #name = "amit" ✗

1st_name = "amit" ✗

first_Name = "amit" ✓

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"

total # blocks = 50

starting address = 0x1

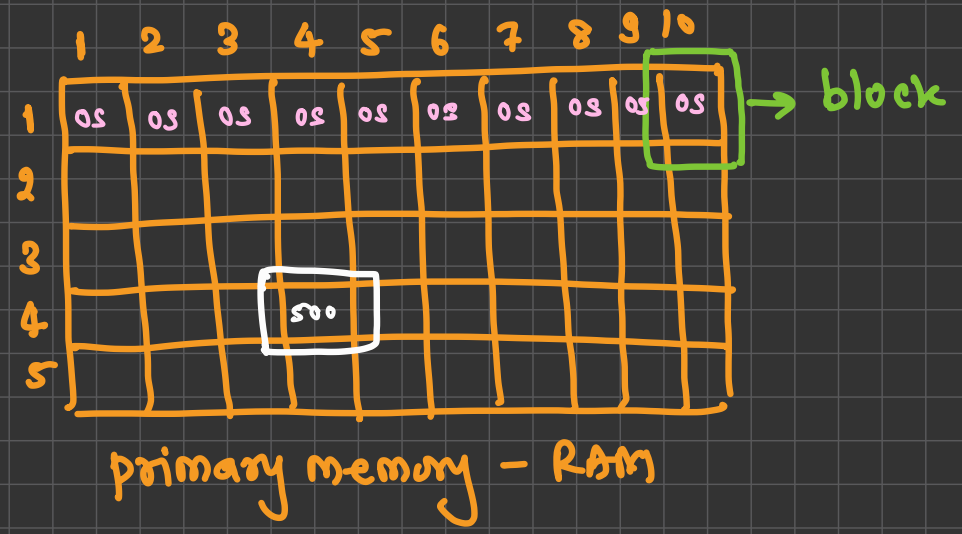
block size = 4KB

num = 200 ← 0x34H → address

variable

↳ int = 4KB

num = 500



Symbol table → symbol = identifier ⇒ collection of all symbols

Symbol name	value	data type	logical address	Real address
num	500	int	0x02	0x34h

Constants

- 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

Data Types

type of data

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

num = 2 = 00000010

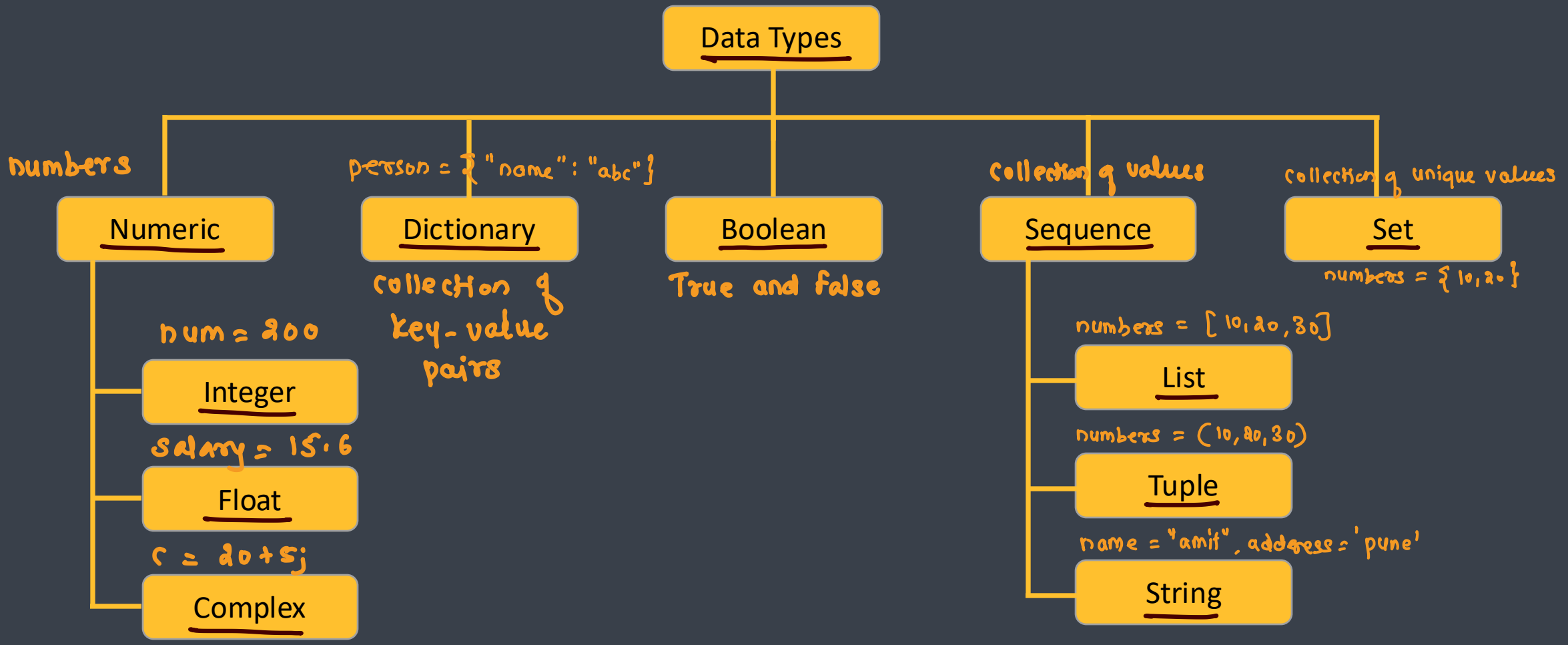
name = "a" = 00100001

Data type is required:

→ to allocate memory

→ to interpret the binary value

Python Data Types



Literals

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

- **Numeric Literals**
  - Numeric Literals are immutable (unchangeable) → Constant
  - 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’

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

Type Hinting

- Due to the dynamic nature of Python, inferring or checking the type of an object being used is especially hard → typem..
- 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

- 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 → automatic type conversion
    - Python automatically converts one data type to another data type
    - This process doesn't need any user involvement
    - E.g.
      - result = 10 + 35.50 = 10.0 + 35.50 = 45.50
  - Explicit Type Conversion
    - 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)

Rule: lower data type (int) gets converted to higher data type (float)
→ there must not be any data loss

(10.35) / float = 10 / int (0.35 gets lost)

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

① Compiler compiles code

② Compiler while building symbol table, always assigns logical address to every identifier

num = 20 ⇒ num = 0x20

③ When application runs, the OS gives real/physical address for every logical address

num = 0x20 = 0x500

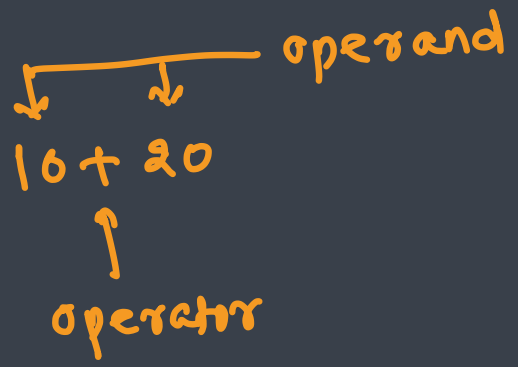
logical	real
Compilation	Execution

Operators

Operators

+ → addition / concatination

- 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



binary operator → operator that takes two operands

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 → int value	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 result

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

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

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