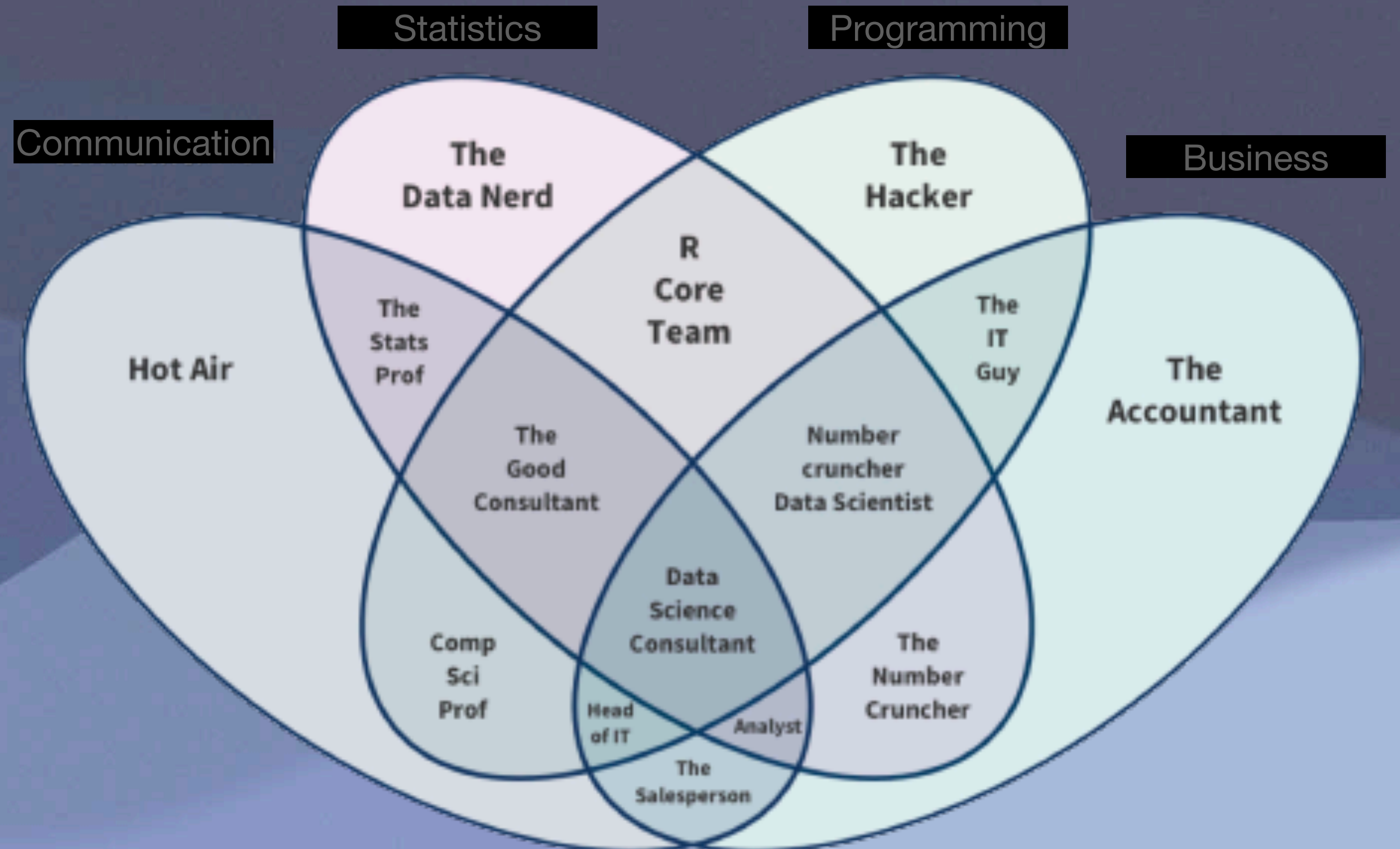


Data Management for Data Science

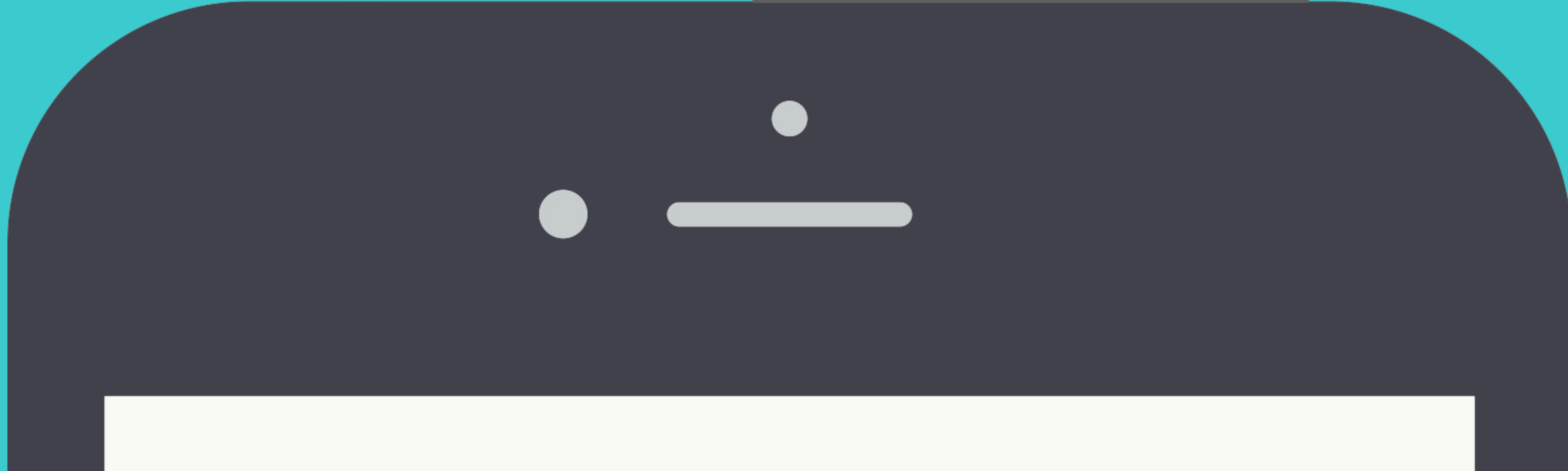
Recitation 1

Anamika Lochab



DATA IS THE NEW OIL

01010101
0101
01 101
10101
01010101



Driven by data

Companies are using data to personalize their products and services. For example, Netflix uses data to recommend movies and TV shows to its users, and Amazon uses data to recommend products to its customers.

Companies are using data to improve their operations. For example, airlines are using data to track flight delays and improve their scheduling, and hospitals are using data to improve patient care.

Governments are using data to make better decisions. For example, public health agencies are using data to track diseases and develop interventions, and transportation agencies are using data to improve traffic flow.

Key challenges in data management for data science

- **Data volume:** The amount of data that is generated and collected today is enormous. Data scientists need to be able to store and process this data efficiently.
- **Data variety:** Data comes in many different formats, from structured data in databases to unstructured data in text documents and images. Data scientists need to be able to work with all of these different types of data
- **Data quality:** Data can be noisy and incomplete. Data scientists need to be able to clean and prepare the data before they can analyze it.

Data Science ≠ Databases

Databases	Data Science
Querying the past	Querying the future

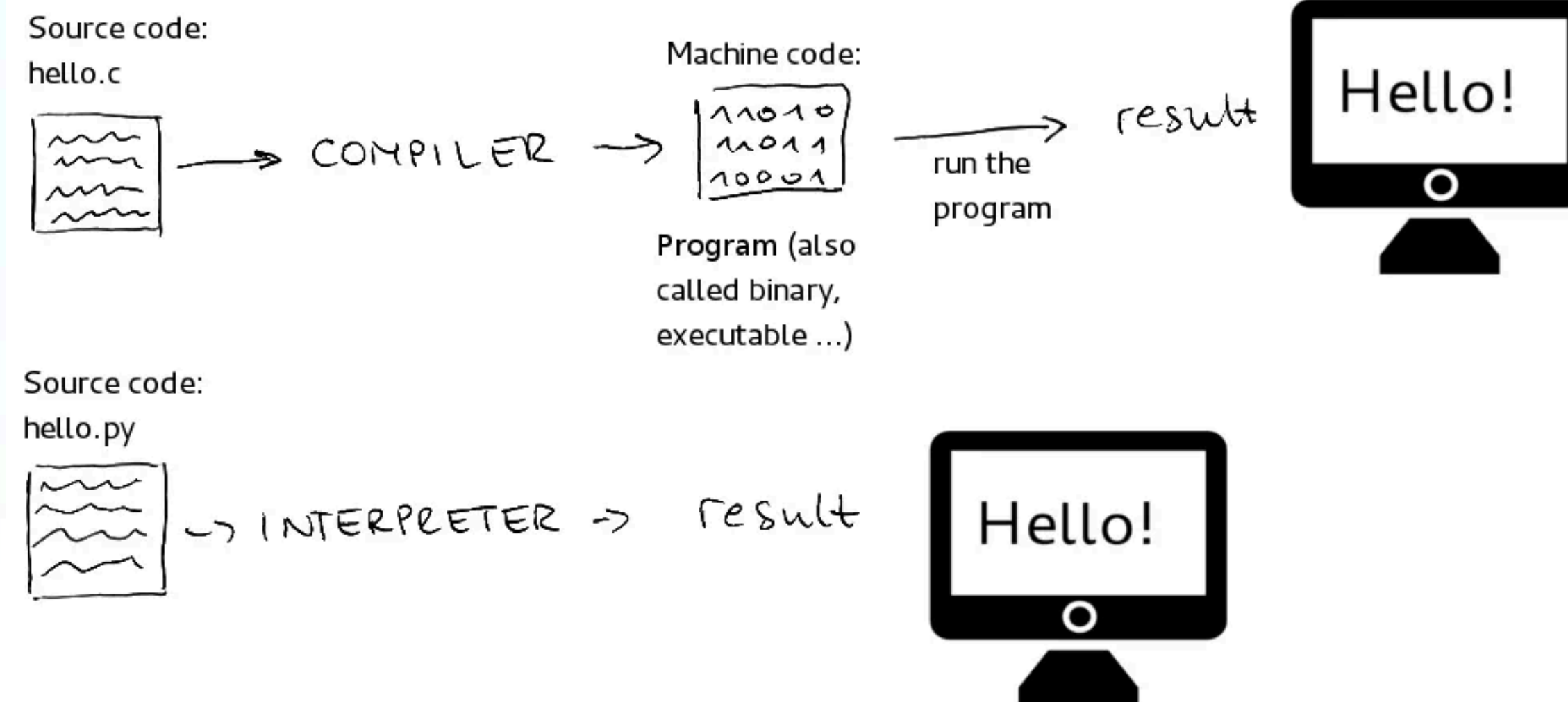
What is Python?

- **Interpreted Language:** Python is an interpreted language, meaning you don't have to compile it before running, which makes the development process quicker.
- **Object-Oriented:** It supports object-oriented programming, allowing for cleaner and more organized code.
- **High-Level:** Python is a high-level language, meaning it's designed to be easy for humans to read and write.
- **Dynamic Semantics:** Python's typing and binding are dynamic, offering flexibility during code execution.
- **Built-In Data Structures:** It has powerful built-in data structures like lists, dictionaries, and sets, making it easier to handle data.
- **Readability:** The language is designed with readability in mind, which makes it easier to understand and maintain.
- **Modularity:** Python supports modules and packages, encouraging code reuse and modular design.
- **Extensive Library:** Python has a broad standard library, allowing you to perform many tasks without needing external libraries.
- **Free to Use:** Python is open-source and free to use and distribute.
- **Platform Independent:** Python works on all major operating systems.

What is Python?

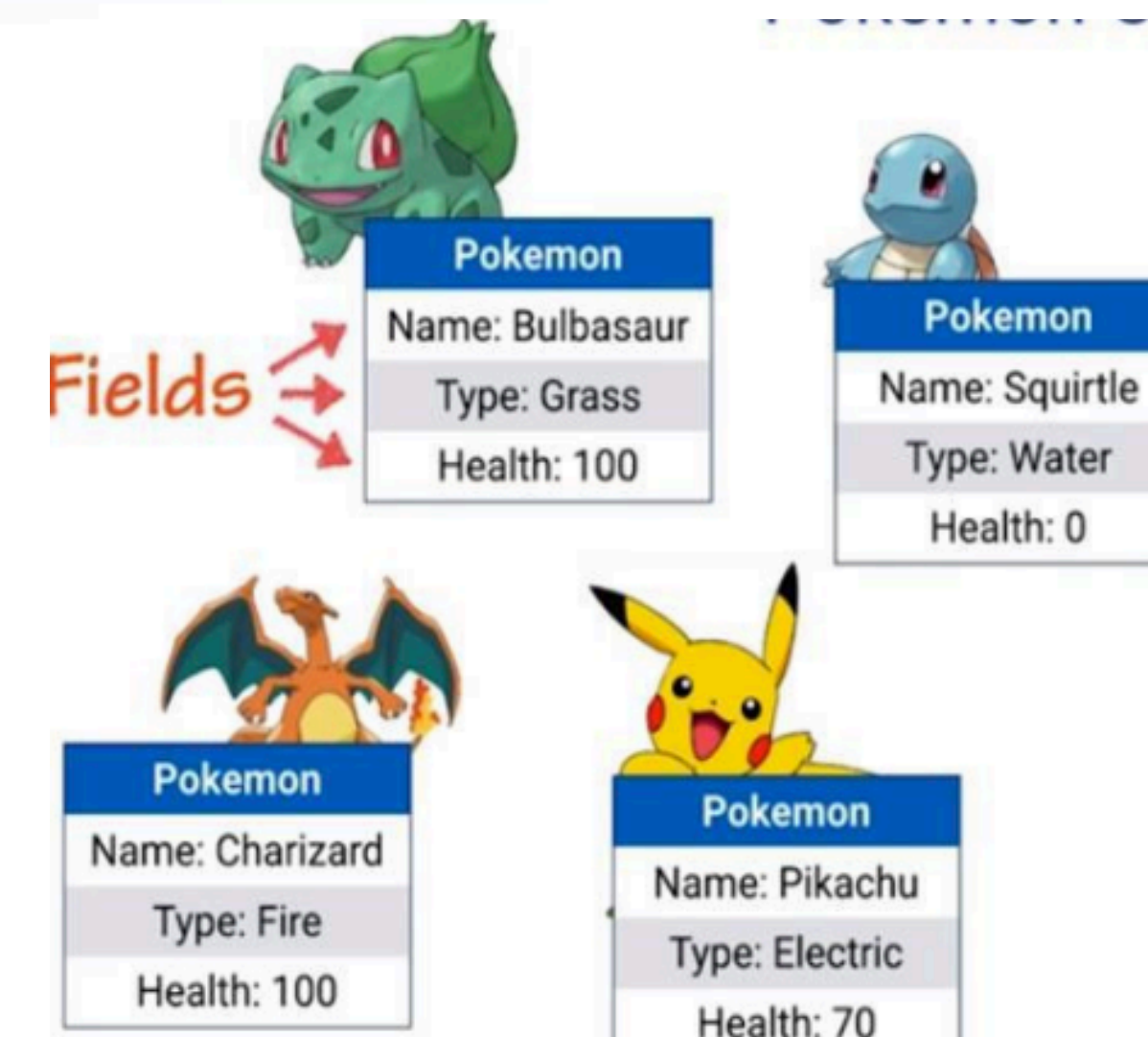
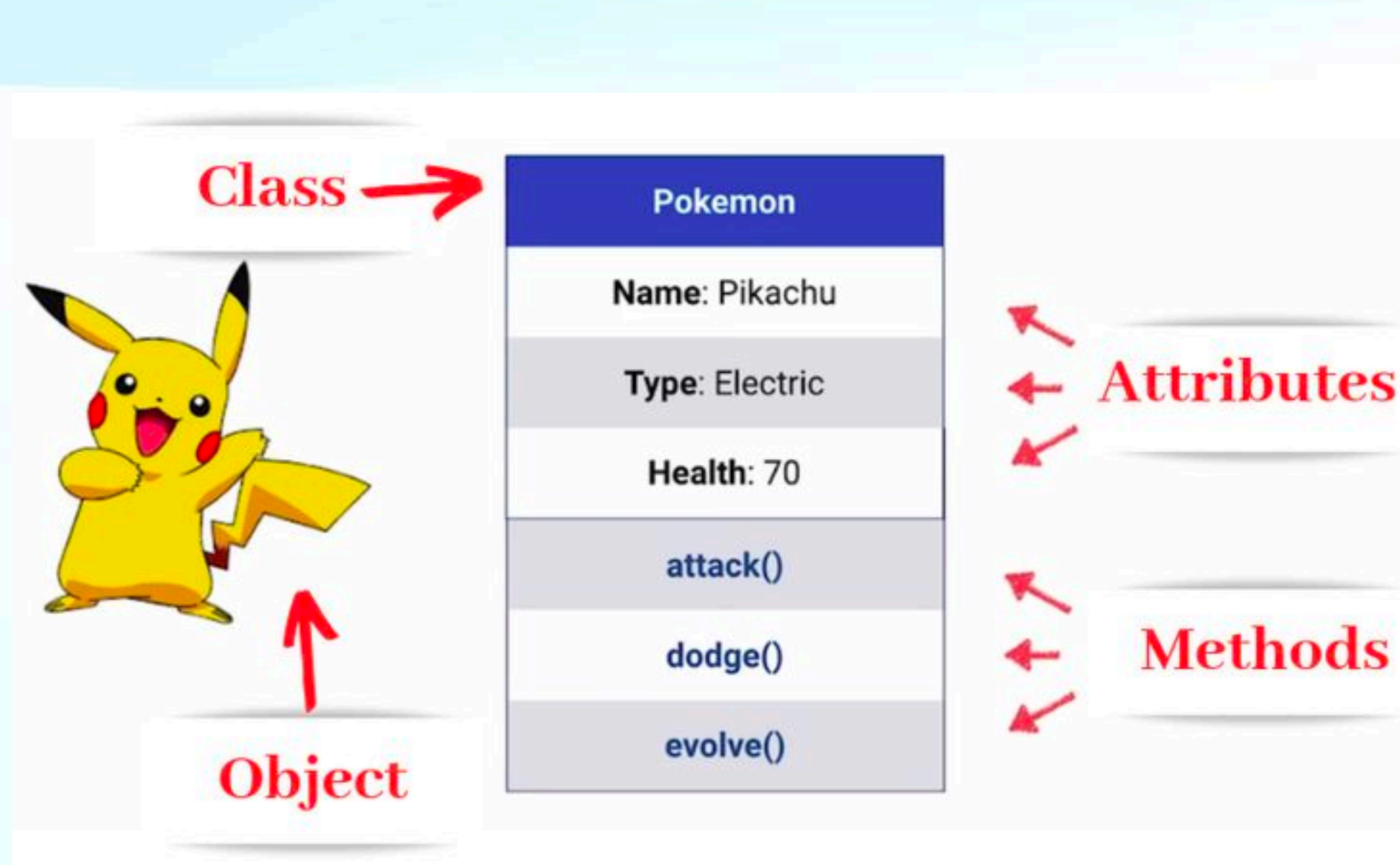
Interpreted Language: Python is an interpreted language, meaning you don't have to compile it before running, which makes the development process quicker.

In some programming languages, you have to perform an extra step called "**compiling**" to make your code runnable. With Python, you don't have to do this. You write your code, and you can run it immediately. This saves time and makes it easier to see if your code works.



What is Python?

- **Object-Oriented:** It supports object-oriented programming, allowing for cleaner and more organized code.
- Imagine your code is like a factory. In a factory, it's easier to manage things if they're sorted into categories or boxes. Object-oriented programming allows you to create these "boxes" (known as objects), which can hold both data and functions. This makes your code easier to read, understand, and maintain.



What is Python?

- **High-Level:** Python is a high-level language, meaning it's designed to be easy for humans to read and write.
- Some programming languages require you to write a lot of complicated code to perform simple tasks. Python is a high-level language, meaning it removes a lot of this complexity. You can do a lot with just a few lines of code, which makes it easier to learn and faster to write programs.
- **Dynamic Semantics:** Python's typing and binding are dynamic, offering flexibility during code execution.
- In some languages, you have to explicitly say what type of data (like a number, word, or list) you'll be using. Python figures this out for you as you go. This offers more freedom and flexibility but requires you to be careful to avoid errors that can arise from this flexibility.

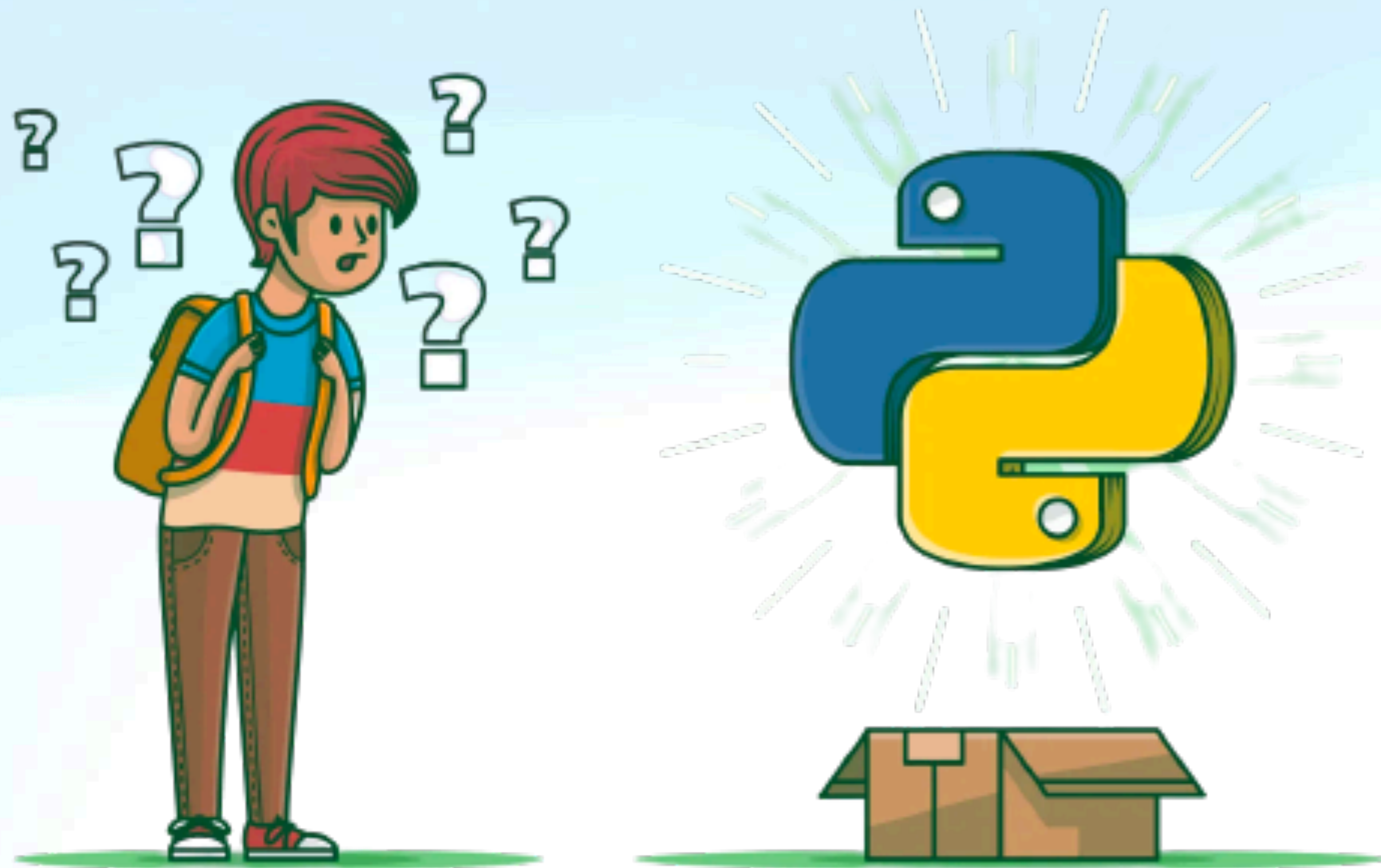
What is Python?

- **Interpreted Language:** Python is an interpreted language, meaning you don't have to compile it before running, which makes the development process quicker.
- **Object-Oriented:** It supports object-oriented programming, allowing for cleaner and more organized code.
- **High-Level:** Python is a high-level language, meaning it's designed to be easy for humans to read and write.
- **Dynamic Semantics:** Python's typing and binding are dynamic, offering flexibility during code execution.
- **Built-In Data Structures:** It has powerful built-in data structures like lists, dictionaries, and sets, making it easier to handle data.
- **Readability:** The language is designed with readability in mind, which makes it easier to understand and maintain.
- **Modularity:** Python supports modules and packages, encouraging code reuse and modular design.
- **Extensive Library:** Python has a broad standard library, allowing you to perform many tasks without needing external libraries.
- **Free to Use:** Python is open-source and free to use and distribute.
- **Platform Independent:** Python works on all major operating systems.

Why was Python created ?

- **Quick to Create:** One of Python's main goals is to make it fast and easy to create new software and applications.
- **Helps with Tasks:** Python can help to automate tasks and link different software together.
- **For Everyone:** Python was designed to help people code faster, whether they are beginners or experts.
- **Friendly for Fixes:** If something goes wrong with your Python program, it's usually easy to figure out the problem and fix it.
- **Speed:** You can write and test code really quickly, so you can see if your idea works in less time.
- **Many Uses:** Python is like a Swiss Army knife for coding. You can use it for web pages, games, data analysis, artificial intelligence, and much more.

The Basics



Real Python



Learn on Collab

What we have learned :

- Assignment uses `=` and comparison uses `==`.
- For numbers `+` `-` `*` `/` `%` are as expected.
 - Special use of `+` for string concatenation.
 - Special use of `%` for string formatting (as with `printf` in C)
- Logical operators are words (`and`, `or`, `not`) not symbols
- The basic printing command is `print`.
- The first assignment to a variable creates it.
 - Variable types don't need to be declared.
 - Python figures out the variable types on its own.
- Python will let you override builtins and keywords

Whitespace

"PYTHON INDENTATION"

(CODE THAT WORKS)

```
n = [3, 5, 7]

def double_list(x):

    for i in range(0, len(x))
      x[i] = x[i] * 2
    return x

print double_list(n)
```

(CODE THAT FAILS)


```
n = [3, 5, 7]

def double_list(x):

    for i in range(0, len(x))
      x[i] = x[i] * 2
    return x

print double_list(n)
```



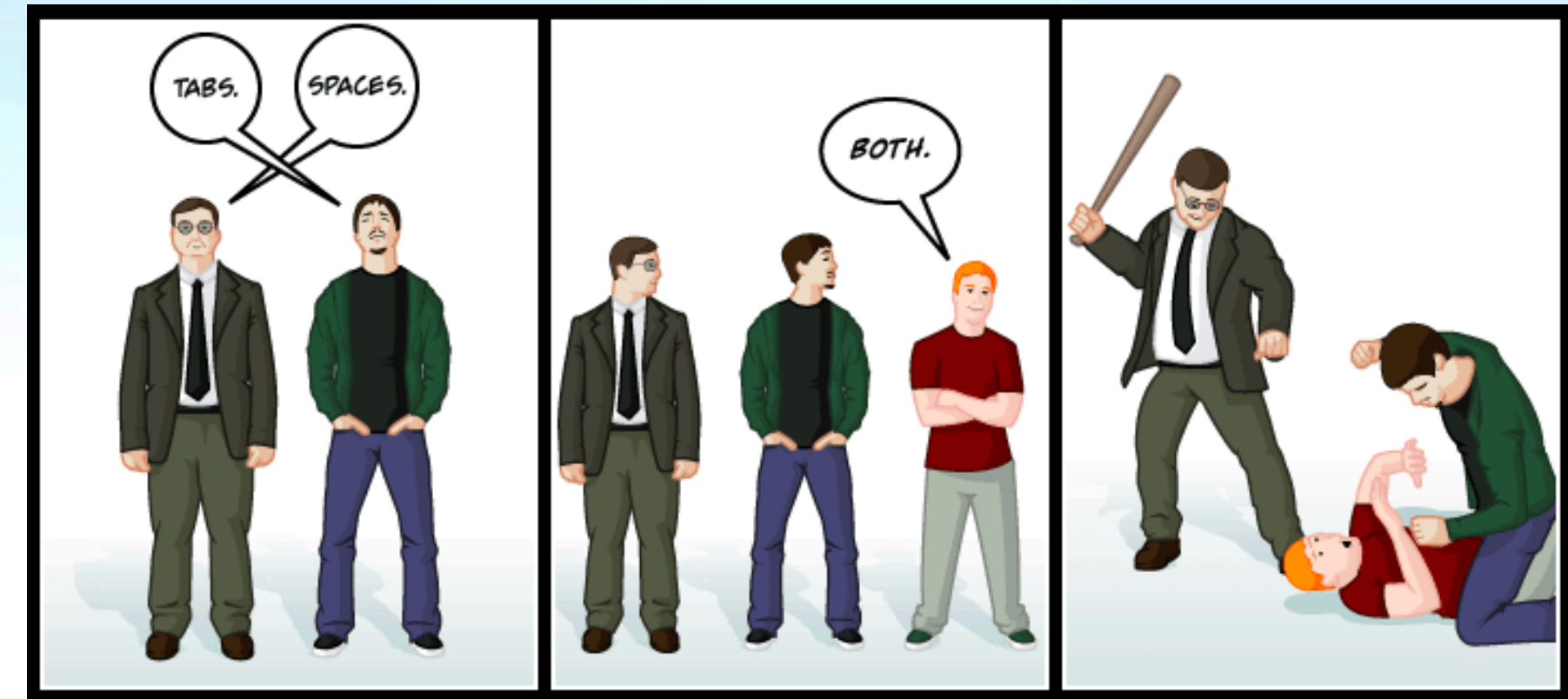
 [HTTPS://TAPAS.IO/SERIES/GRUMPY-CODES](https://tapas.io/series/grumpy-codes)



 CARDBOARDVOICE

White Space

- **Whitespace is meaningful in Python: especially indentation and placement of newlines.**
- **Use a newline to end a line of code.**
 - Use `\` when must go to next line prematurely.
- **No braces { } to mark blocks of code in Python.**
 - Use consistent indentation instead.
 - The first line with less indentation is outside of the block.
 - The first line with more indentation starts a nested block
- **Often a colon appears at the start of a new block. (E.g. for function and class definitions.)**



Understanding Reference Semantics

- Assignment manipulates references
 - `x = y` does not make a copy of the object `y` references
 - `x = y` makes `x` reference the object `y` references

```
a = [1, 2, 3] # Mutable type
b = a

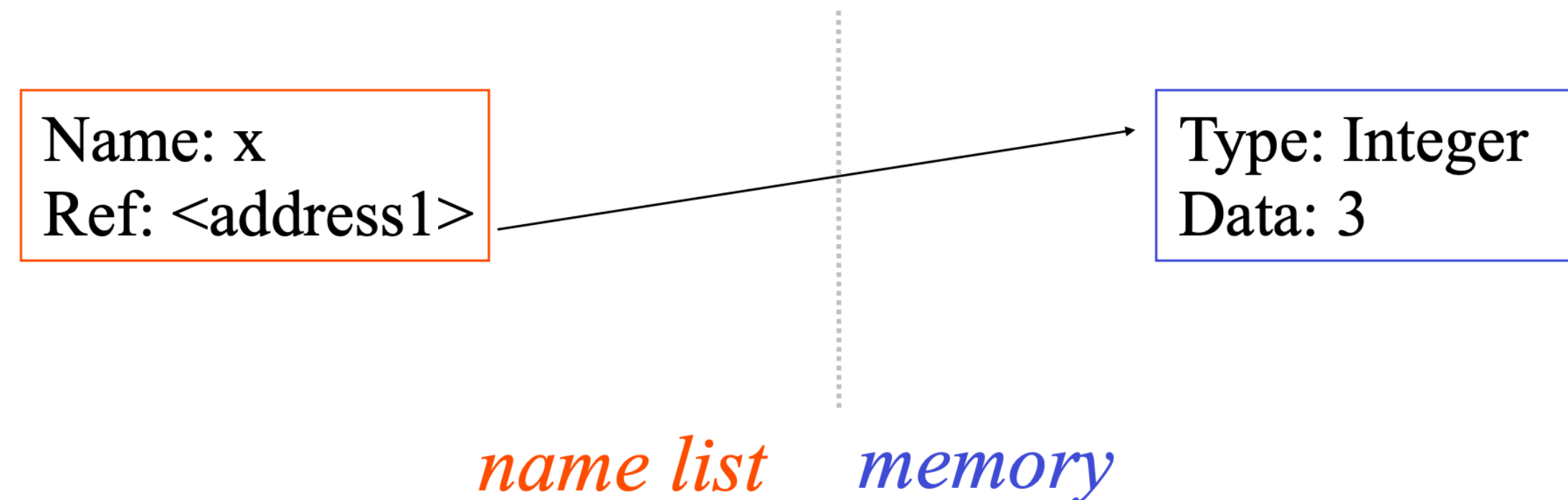
b.append(4)

print(a, "\t\t", b) # Output will be [1, 2, 3, 4]
```

☞ [1, 2, 3, 4]

[1, 2, 3, 4]

- There is a lot going on when we type: **x = 3**
- First, an integer **3** is created and stored in memory
- A name **x** is created
- An **reference** to the memory location storing the **3** is then assigned to the name **x**
- So: When we say that the value of **x** is **3**
- We mean that **x** now refers to the integer **3**



- What will be the value of x?

```
x = 3          # Creates 3, name x refers to 3
y = x          # Creates name y, refers to 3.
y = 4          # Creates ref for 4. Changes y.
print x        # No effect on x, still ref 3.
```

- So, for simple built-in datatypes (integers, floats, strings), assignment behaves as you would expect.

- For other data types (**lists, dictionaries, user-defined types**), assignment works differently.
- These datatypes are “**mutable**.”
- When we change these data, we do it in place.
- We don't copy them into a new memory address each time.
- If we type `y=x` and then modify `y`, both `x` and `y` are changed.

immutable

```
>>> x = 3
>>> y = x
>>> y = 4
>>> print x
3
```

mutable

```
x = some mutable object
y = x
make a change to y
look at x
x will be changed as well
```


Sequence Types

In python, sequence types are data structures that can hold multiple items in an ordered collection. They are incredibly flexible and form the basis for many data manipulation tasks. The primary sequence types in Python are:

- **Tuple :**
 - A simple immutable ordered sequence of items
 - Items can be of mixed types, including collection types
 - Defined using parentheses () or just commas.
- **Strings :**
 - Immutable
 - Conceptually very much like a tuple
 - Defined using quotes “” or ‘’
- **List :**
 - Mutable ordered sequence of items of mixed types
 - Defined using square brackets [].
- **Range :**
 - Immutable, The numbers in a range are ordered.
 - Defined using the **range()** function.



Learn on Collab

Lambda Function

In Python, a lambda function is a small anonymous function, which means it's a function without a name. Lambda functions are often used for short, simple operations that can be defined in a single line of code.

```
lambda arguments: expression
```

```
def func(args):  
    return ret_val
```

is equivalent to:

```
= lambda :
```