



# Data Science with Python

By Vaishali KUNJIR

# About Me



## Vaishali KUNJIR

Lead Data Engineer | Data Science R & D Engineer |  
Senior software Engineer | Full stack developer(Angular,  
Python) | Automation | Django | Flask | Frontend |  
Python expert | Scrum Master| project manager | Fluent  
French speaker

### Education:

Master in Machine learning specialization in automatic  
image and natural language processing

Université de Caen Normandie, Caen, France

Master of Computer Science

Savitribai Phule Pune University, Pune, India

### CERTIFICATION :

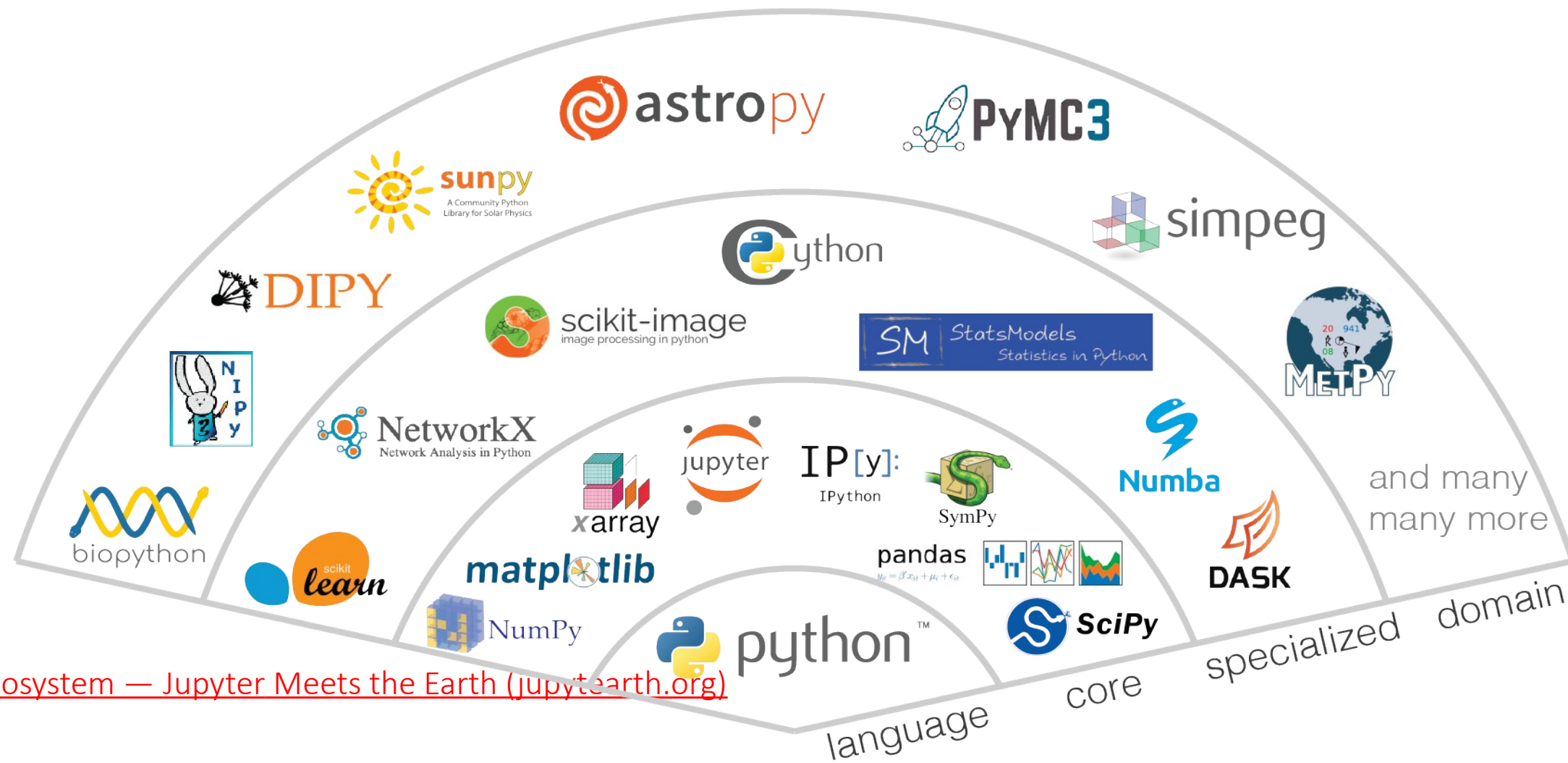
Agile with Atlassian Jira MOOC Coursera

Atlassian Community Scrum master certification

Machine learning MOOC Coursera, Stanford University

Advanced Diploma in software engineering

CAT, Pune, India FLE A2 French language



\*\* Reference: Ecosystem — Jupyter Meets the Earth (jupyterearth.org)

# Introduction to Python

- What is python?
- Scope of Python language (Job opportunities and companies)
- Setup of Development environment
- Basic Syntax, Comments, Variables, Data Types, Operators, Decision Making, Loops, Numbers, Data Structures (Numbers, String, Lists, Tuples, Dictionary, Date & Time), Functions, Modules, File I/O, Exceptions
- More information about python, Please refer ➔ [The Python Tutorial — Python 3.11.4 documentation](#)



# What is python?

**Python** is a very popular general-purpose interpreted, interactive, object-oriented, and high-level programming language. Python is dynamically-typed and garbage-collected programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL)

## [Why to learn Python?](#)

- Python is Open Source which means its available free of cost.
- Python is simple and so easy to learn
- Python is versatile and can be used to create different web applications, AI/ML algorithms, Data analysis, Scripts etc.
- Python has powerful development libraries include AI, ML etc.
- Python is much in demand and ensures high salary

## [Characteristics of Python](#)

Following are important characteristics of **Python Programming** –

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.
- Python has been ported to the Java and .NET virtual machines

# Scope of Python language (Job opportunities and Companies)

## ➤ Job Opportunities

- Game developer
- Web designer
- Python developer
- Full-stack developer
- Machine learning engineer
- Data scientist
- Data analyst
- Data engineer
- DevOps engineer
- Software engineer
- Many more other roles



## ➤ Companies

- Google
- Intel
- NASA
- PayPal
- Facebook
- IBM
- Amazon
- Netflix
- Pinterest
- Uber
- Many more...

# Setup of Development environment

Python is available on a wide variety of platforms. Following is the list of platforms which are compatible with python:

- Unix (Solaris, Linux, FreeBSD, AIX, HP/UX, SunOS, IRIX, etc.)
- Linux
- Windows
- Mac OS X
- Win 9x/NT/2000
- Macintosh (Intel, PPC, 68K)
- OS/2
- DOS (multiple versions)
- PalmOS
- Nokia mobile phones
- Windows CE
- Acorn/RISC OS
- BeOS
- Amiga
- VMS/OpenVMS
- QNX
- VxWorks
- Psion

# Setup of Development environment

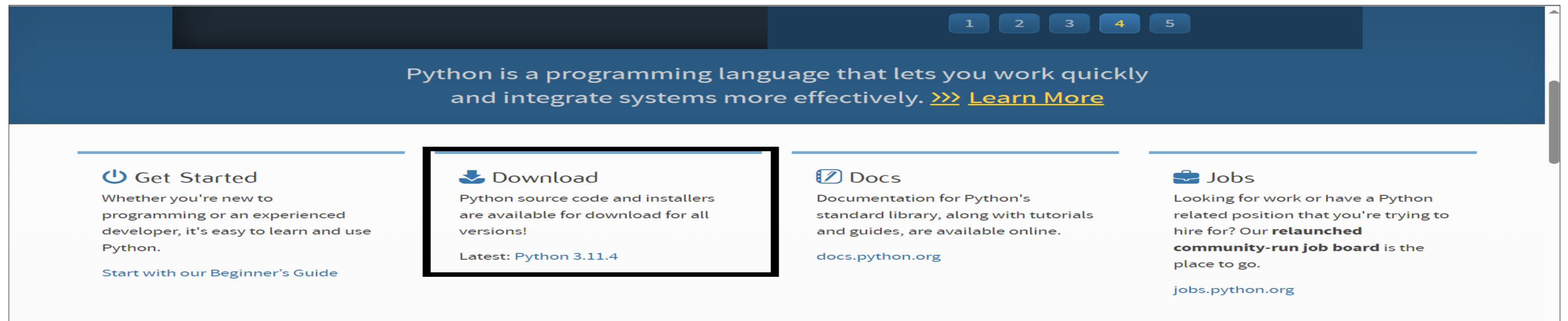
- Open command prompt, type **python**. If you see following output, then it means that, python is installed.

C:\> Command Prompt - python

```
Microsoft Windows [Version 10.0.22000.1098]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vaish>python
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- If no output is displayed, means python installation is required
- Go to [www.python.org](https://www.python.org) → Downloads



The screenshot shows the Python.org homepage. At the top, a dark blue banner contains the text: "Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)". Below the banner, there are five navigation buttons: 1, 2, 3, 4, and 5. The 'Download' button (4) is highlighted. The main content area is divided into four columns. The first column is titled 'Get Started' and contains the text: "Whether you're new to programming or an experienced developer, it's easy to learn and use Python. Start with our [Beginner's Guide](#)". The second column is titled 'Download' and contains the text: "Python source code and installers are available for download for all versions! Latest: [Python 3.11.4](#)". This column is highlighted with a black border. The third column is titled 'Docs' and contains the text: "Documentation for Python's standard library, along with tutorials and guides, are available online. [docs.python.org](#)". The fourth column is titled 'Jobs' and contains the text: "Looking for work or have a Python related position that you're trying to hire for? Our **relaunched community-run job board** is the place to go. [jobs.python.org](#)".



# Setup of Development environment

## Files

| Version   | Operating System | Description              | MD5 Sum                          | File Size | GPG                 | Sigstore                  |
|---|------------------|--------------------------|----------------------------------|-----------|---------------------|---------------------------|
| <a href="#">Gzipped source tarball</a>              | Source release   |                          | bf6ec50f2f3bfa6ffbdb385286f2c628 | 26526163  | <a href="#">SIG</a> | <a href="#">.sigstore</a> |
| <a href="#">XZ compressed source tarball</a>        | Source release   |                          | fb7f7eae520285788449d569e45b6718 | 19954828  | <a href="#">SIG</a> | <a href="#">.sigstore</a> |
| <a href="#">macOS 64-bit universal2 installer</a>   | macOS            | for macOS 10.9 and later | 91498b67b9c4b5ef33d1b7327e401b17 | 43120982  | <a href="#">SIG</a> | <a href="#">.sigstore</a> |
| <a href="#">Windows embeddable package (32-bit)</a> | Windows          |                          | 81b0acfcdd31a73d1577d6e977acbd6  | 9596761   | <a href="#">SIG</a> | <a href="#">.sigstore</a> |
| <a href="#">Windows embeddable package (64-bit)</a> | Windows          |                          | d0e85bf50d2adea597c40ee28e774081 | 10591509  | <a href="#">SIG</a> | <a href="#">.sigstore</a> |
| <a href="#">Windows embeddable package (ARM64)</a>  | Windows          |                          | bdce328de19973012123dc62c1cfa7e9 | 9965162   | <a href="#">SIG</a> | <a href="#">.sigstore</a> |
| <a href="#">Windows installer (32 -bit)</a>         | Windows          |                          | 9ec180db64c074e57bdcca8374e9ded6 | 24238000  | <a href="#">SIG</a> | <a href="#">.sigstore</a> |
| <a href="#">Windows installer (64-bit)</a>          | Windows          | Recommended              | e4413bb7448cd13b437dffffba294ca0 | 25426160  | <a href="#">SIG</a> | <a href="#">.sigstore</a> |
| <a href="#">Windows installer (ARM64)</a>           | Windows          | Experimental             | 60785673d37c754ddceb5788b5e5baa9 | 24714240  | <a href="#">SIG</a> | <a href="#">.sigstore</a> |

# Python Installation with Anaconda/Jupyter Notebook

Learn the benefits of using Anaconda and jupyter Notebook for Python installation. Simplified process, easy package management, and integrated development environment



**by vaishali kunjir**



# N PROGRAM

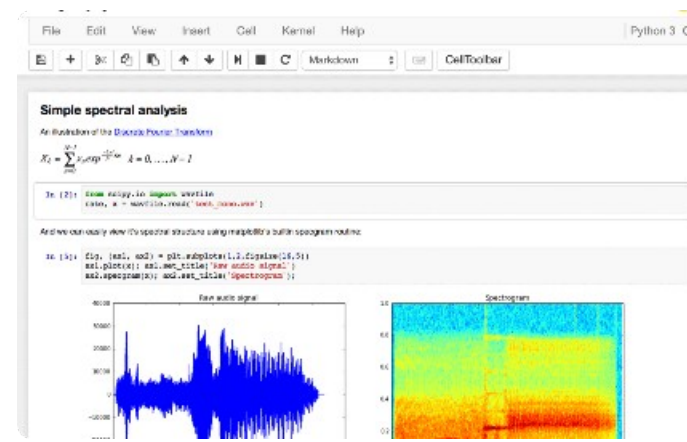
# Introduction to Anaconda

Get acquainted with Anaconda, the Python distribution that provides a comprehensive package manager and environment management system.



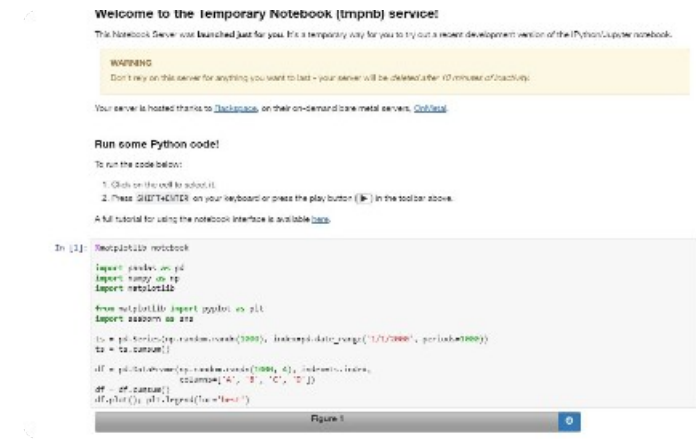
## What is Anaconda?

Learn about Anaconda's features, including pre-installed packages and tools for data science.



## What is Jupyter Notebook?

Discover the interactive coding interface that jupyter Notebook offers, enabling documentation and computational elements.



## Jupyter Notebook Applications

Explore the various applications of jupyter Notebook, such as data exploration, visualization, and machine learning.

# Step-by-Step Installation Guide

## Downloading Anaconda

Access the official Anaconda website and obtain the installation file for your operating system.

## Installing Anaconda

Follow the installation wizard instructions to set up Anaconda on your computer.

## Setting up the Environment

Configure the Anaconda environment variables to ensure seamless integration with your system.

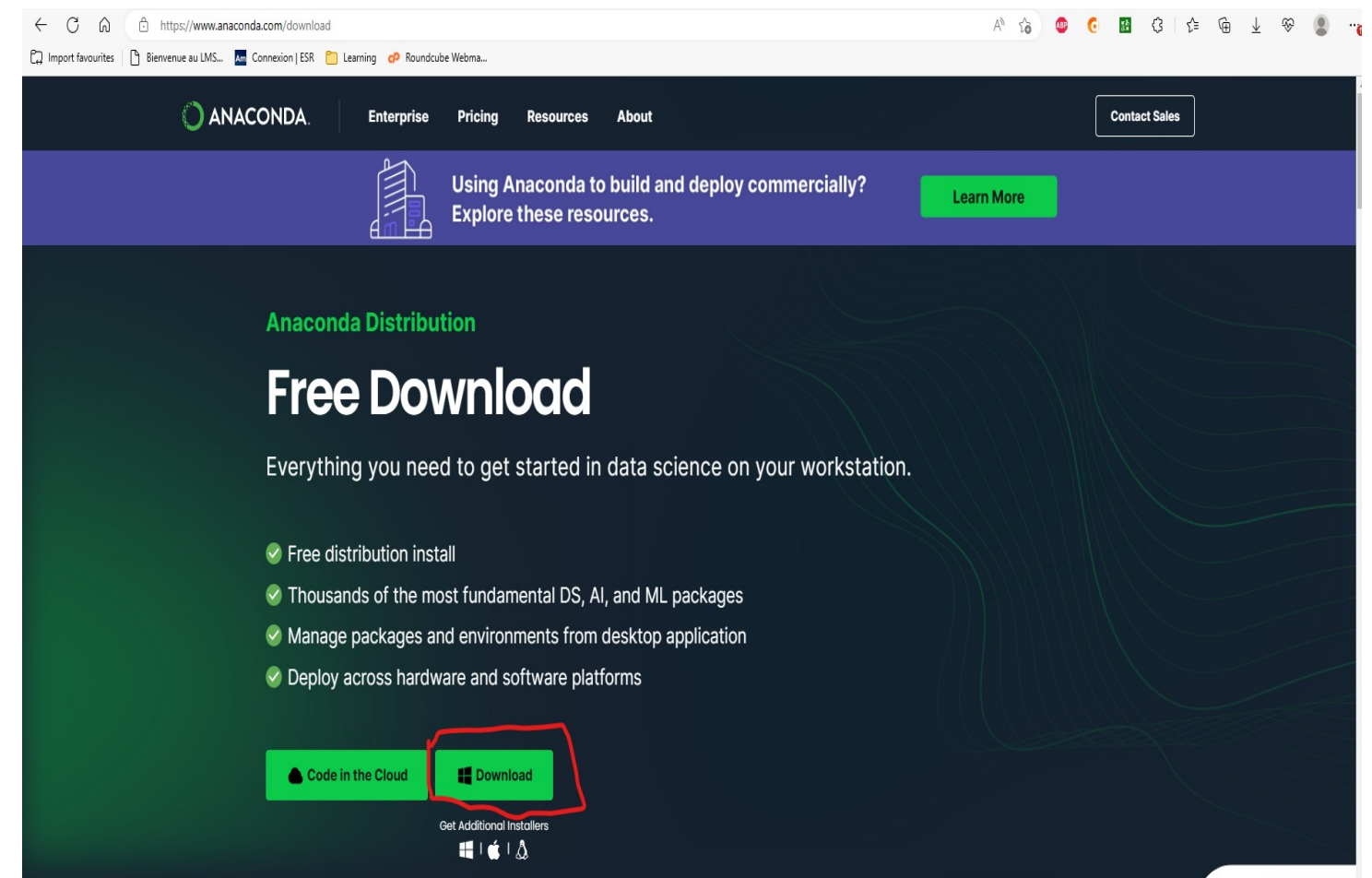
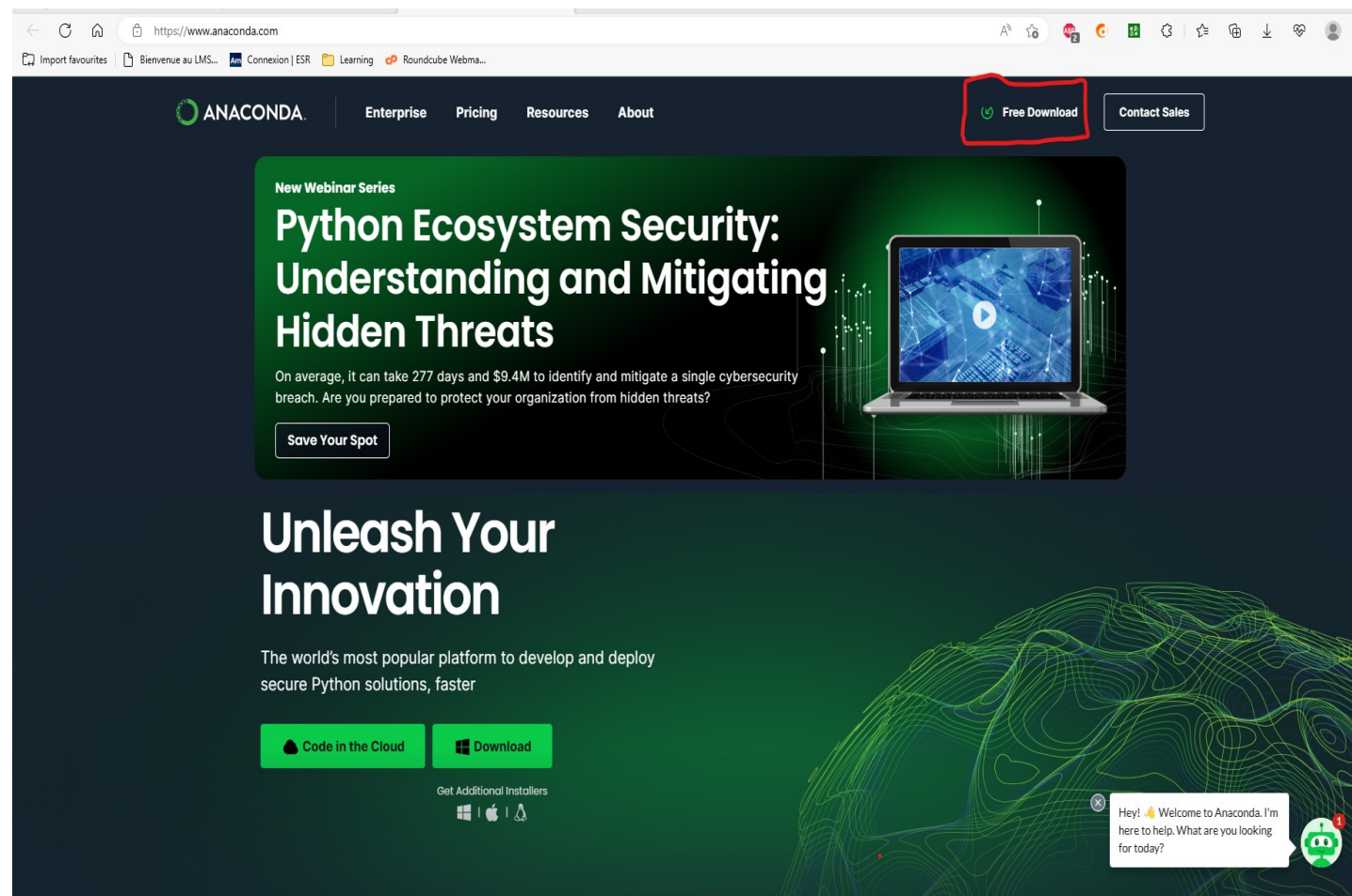
## Verifying the Installation

Confirm that Anaconda has been installed correctly by running a simple Python command in the terminal.



# Anaconda installation

- Browse website <https://www.anaconda.com/> and click on **Free Download**. Click on **Download** button to download Anaconda distribution.
- Refer this installation guide <https://docs.anaconda.com/free/anaconda/install/index.html>



# Exploring Jupyter Notebook

1

## Opening Jupyter Notebook

Learn how to launch Jupyter Notebook from the command line or Anaconda Navigator.

2

## Creating and Running Scripts

Discover how to create and execute Python scripts within the Jupyter Notebook environment.

3

## Saving and Sharing Work

Understand how to save your Jupyter Notebook projects and share them with others as interactive documents.



# Conclusion and Next Steps

## 1 Recap of the Installation Process

Download and install Anaconda

## 2 Further Resources

<https://learning.anaconda.cloud/jupyter-notebook-basics>



# Variables, Data Types, Operators and Operands

Welcome to the world of Python! In this presentation, we'll explore the magic of variables, data types, operators and operands in Python and learn how to create beautiful programs as if they were a piece of art.

# What are Variables in Python?

**1**

## Definition

Variables are like containers that hold data in a computer program.

**2**

## Importance

They enable us to store and manipulate data in a program, making it more flexible and powerful.

**3**

## Declaration

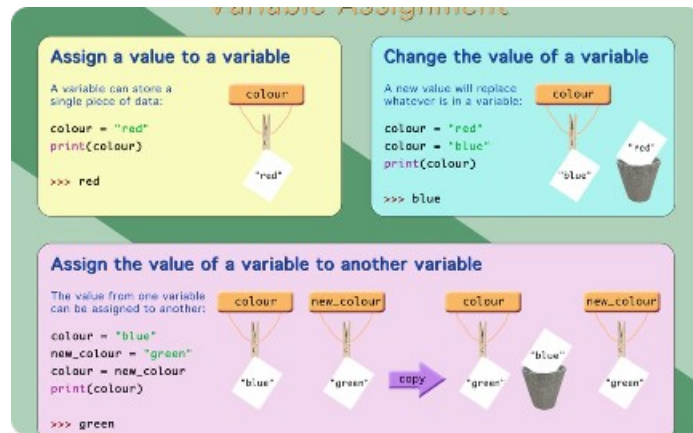
We declare variables by assigning a name to them using the syntax "variable\_name = value".

**4**

## Assignment

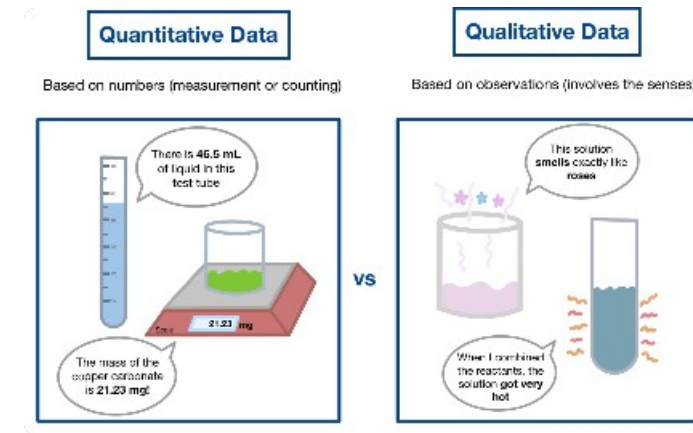
We assign values to variables using the equals (=) sign.

# Manipulating Variables in Python



The screenshot shows a spreadsheet with the following data:

|    | A        | B      | C               | D  | E |
|----|----------|--------|-----------------|--|---|
| 1  | Base     | Second | Combined        | Formula                                      |   |
| 2  | Base     | ball   | Baseball        | <code>=CONCATENATE(A2,B2)</code>             |   |
| 3  | Base     | ball   | Baseball        | <code>=A3&amp;B3</code>                      |   |
| 4  | Mary     | Jones  | Mary Jones      | <code>=CONCATENATE(A4, " ", B4)</code>       |   |
| 5  | Joe      | Smith  | Joe Smith       | <code>=A5&amp;" "&amp;B5</code>              |   |
| 6  | Thompson | Holt   | Thompson & Holt | <code>=CONCATENATE(A6, " &amp; ", B6)</code> |   |
| 7  | 123      | 456    | 123456          | <code>=A7&amp;B7</code>                      |   |
| 8  |          |        |                 |  |   |
| 9  |          |        |                 |  |   |
| 10 |          |        |                 |  |   |
| 11 |          |        |                 |  |   |
| 12 |          |        |                 |  |   |
| 13 |          |        |                 |  |   |



## Basic Operations

Variables can be added, subtracted, multiplied, and divided just like numbers.

## Concatenation

Variables that store strings can be combined using the "+" operator.

## Comparisons

Variables can be compared using logical operators like "=", "!=", "<", ">" and "<=", ">=".

# Scope and Lifetime of Variables

## Scope

Variables have a scope, which determines where they can be accessed in a program.

For example, a variable declared inside a function can only be accessed from within the function.

## Lifetime

The lifetime of a variable is the period during which it exists in memory while the program is running. Once a variable goes out of scope, it may be deleted by the interpreter to free up memory.

## Global

Global variables are accessible from anywhere in the program and can be useful for storing values that need to be accessed in multiple places.

## Local

Local variables are only accessible from within the block of code in which they are declared.



# Best Practices for Naming Variables

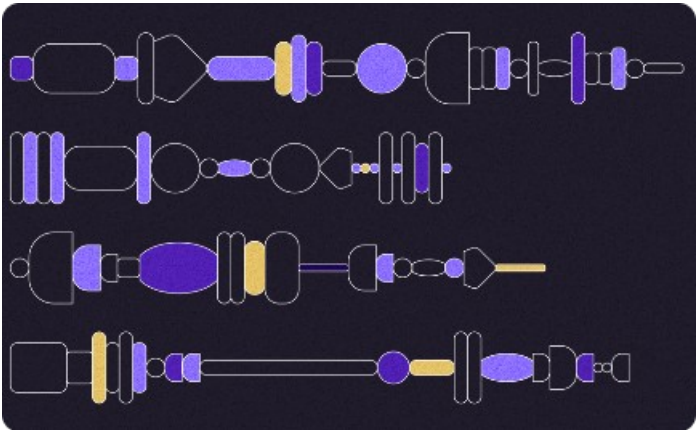
|                |            | capitalized   |                     |
|----------------|------------|---|---------------------|
| Database table | plural     | lowercase with underscores separating words                     | book_clubs          |
| Controller     | pluralized | first letter of each word capitalized                           | BookClubsController |
| Module         | singular   | first letter of each word capitalized                           | Club                |
| Foreign keys   | singular   | singularized table name followed by id separated by underscores | book_club_id        |
| Primary keys   | id         | automatically generated   | id                  |

## Variable Names

When naming a variable follow these rules and conventions:

- Make the variable name meaningful. That means that **L** is not a meaningful variable name but **length** is meaningful.
- Start all variable names with a lower-case letter.
- Variable names may also contain digits.
- If a variable name is more than one word long, capitalize each of the other words without adding any spaces.

Below are some examples of valid variable names:  
**number sum32 testAverage areaOfTrapezoid**  
and below are some examples of invalid variable names:



## Naming convention

Variables should have meaningful names that reflect their purpose. Use lowercase letters for variable names and use underscores to separate words.

## Meaningful names

Choose names that are descriptive and easy to understand. Avoid abbreviations and overly generic names.

## Length

Keep variable names short and sweet. Long names can be difficult to read and prone to typos.



# Python Reserved Words

The following list shows the Python keywords. These are reserved word's and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

|         |          |          |
|---------|----------|----------|
| and     | as       | assert   |
| break   | class    | continue |
| def     | del      | elif     |
| else    | except   | False    |
| finally | for      | from     |
| global  | if       | import   |
| in      | is       | lambda   |
| None    | nonlocal | not      |
| or      | pass     | raise    |
| return  | True     | try      |
| while   | with     | yield    |

# Data Types: The Two Sides of the Binary World

Python data types include numbers, strings, booleans, lists, tuples, and dictionaries. Each data type has unique functions and properties, and learning to use them effectively is crucial for intricate programming.

## Strings

Strings are sequences of Unicode characters. They are mutable and can be tweaked, manipulated, and concatenated to create new strings or alter existing ones.

## Booleans

Boolean data types represent two values: True or False. They are often used to test conditions and validate if statements.

## Dicts, Lists, Tuples, Set

Python's dictionaries and lists are mutable and can store a collection of any type of information we want. They are incredibly flexible and useful in more complex programs. Tuples are immutable

# Data Types Declaration

**1) Integers (int):** Integers are whole numbers, positive or negative, without any decimal point.

Example:  
`age = 30`

**2) Floating-Point Numbers (float):** Floating-point numbers, or floats, represent real numbers and can have decimal points.

Example:  
`Pi = 3.14`

**3) Strings (str):** Strings are sequences of characters, and they are used to represent text in Python. Strings are enclosed in either single (') or double (") quotes.

Example:  
`name = "Alice"`

**4) Boolean (bool):** Boolean data type represents two values: True and False. Booleans are often used in conditional and loop statements.

Example:  
`is_adult = True`

# Data Types Declaration

**5) Lists (list):** Lists are ordered collections of items, which can be of different data types. Lists are defined by square brackets [ ]

Example:

```
Numbers = [1, 2, 3, 4, 5]
```

**6) Tuples (tuple):** Tuples are similar to lists, but they are immutable, meaning their elements cannot be changed after creation. Tuples are defined by parentheses ( )

Example:

```
Coordinates = (3, 4)
```

**7) Dictionaries (dict):** Dictionaries are collections of key-value pairs. They are unordered and can be used to store data values like a map. Dictionaries are defined by curly braces { }

Example:

```
person = {"name": "Bob", "age": 25, "city": "New York"}
```

**8) Sets:** You can create a set by placing a comma-separated sequence of items inside curly braces {}, or by using the set() constructor.

Example:

```
# Creating a set with curly braces
```

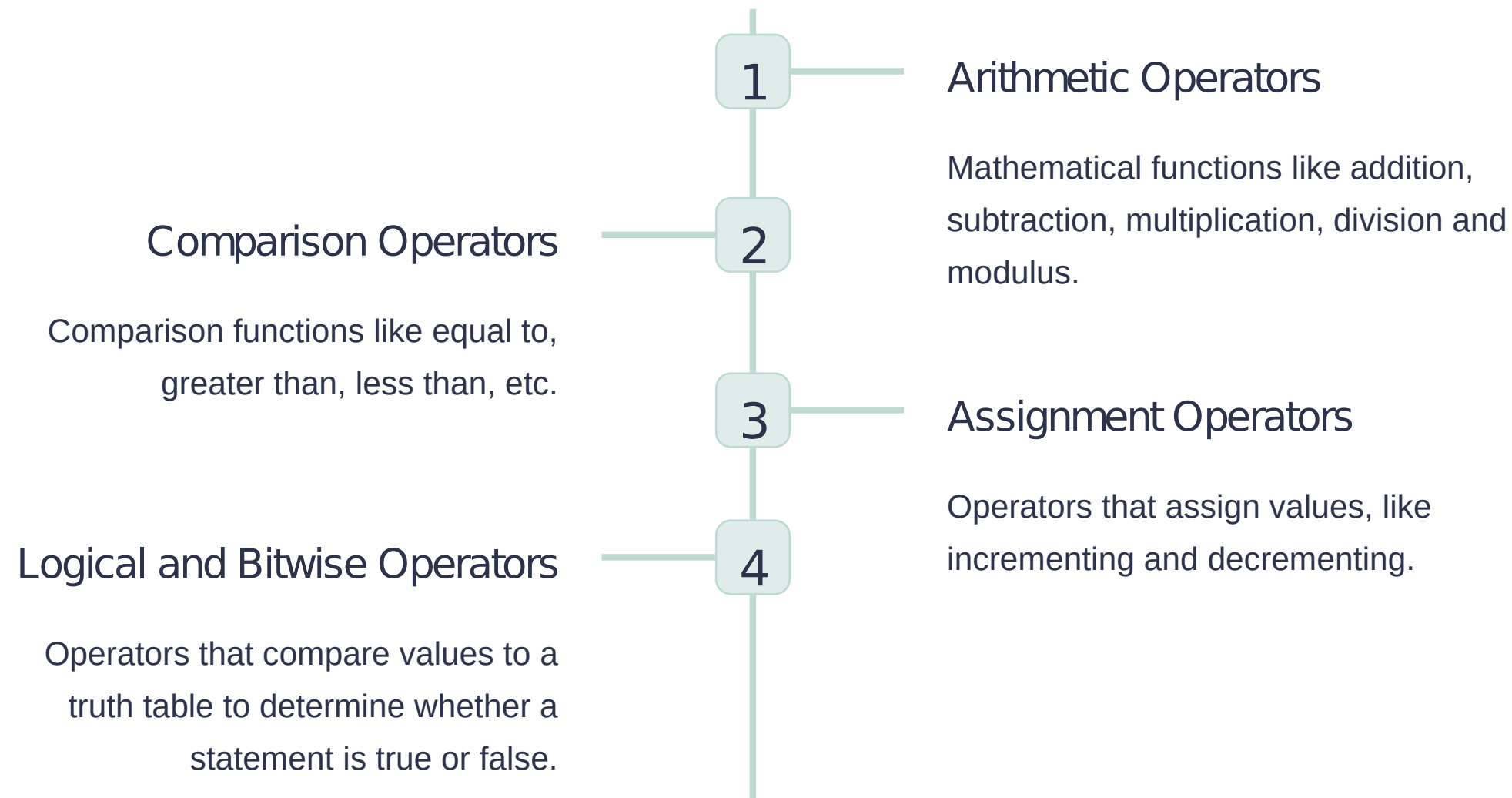
```
my_set = {1, 2, 3, 4, 5}
```

```
# Creating a set with set() constructor
```

```
another_set = set([3, 4, 5, 6, 7])
```

# Operators and Operands: The Rhythm of Your Code

Operators and operands execute functions on data to return a value.



# Identity Operators: Differentiating Between Objects in Memory

```
1 class Employee:
2     count = 0 # class variables
3     ids_list = []
4
5     def __init__(self, i):
6         self.id = i # instance variable
7         Employee.count += 1
8         self.ids_list.append(i)
9
10
11 for x in range(0, 10):
12     emp = Employee(x)
13
14 print(f'Number of employees created = {Employee.count}')
15 print(f'List of employee ids allocated = {Employee.ids_list}')
16
17 emp = Employee(1000)
18 print(f'List of employee ids allocated = {emp.ids_list}')
19
```

Run: class\_examples

/Users/pankaj/Documents/PycharmProjects/AskPython/venv/bin/python /Users/pankaj/Documents/PycharmProjects/AskPython/venv/bin/python

Number of employees created = 10  
List of employee ids allocated = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

is vs. ==

The "is" operator checks for object identity, while the "==" operators check for object similarity.

|    |                          |        |
|----|--------------------------|--------|
| >  | Greater than             | x > y  |
| <  | Less than                | x < y  |
| == | Equal to                 | x == y |
| != | Not equal to             | x != y |
| >= | Greater than or equal to | x >= y |

Identity vs. Equality

The "id()" function serves to identify an object in memory, while the "==" operator checks for equality of values.



# Membership Operators: Getting Inside the List

Membership operators check whether a value exists within a sequence, like a list or string.

`in`

Determines whether the value is present in the object.

`not in`

Determines whether the value is not present in the object.

# Pro Tips for Pythonic Programs

Make your code more efficient and pythonic with these expert tips!



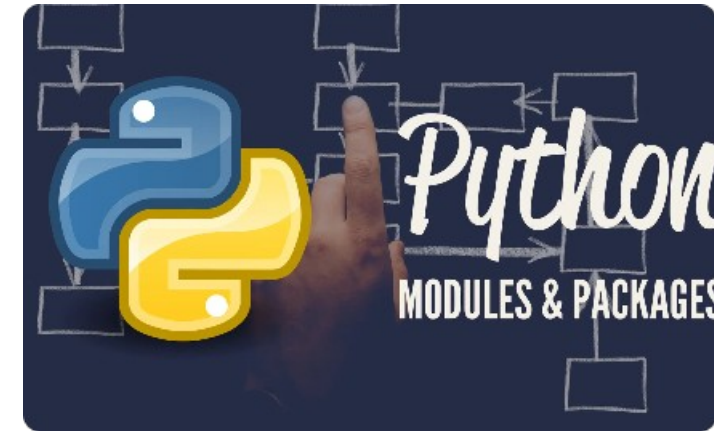
## Clean Code

Keep your code concise and organized, with comments explaining your reasoning.



## Readability

Make sure your code is easy to read, with clear variable names and logical structure.



## Useful Libraries

Python has a vast array of libraries such as Numpy, Pandas, Keras designed specifically to solve complex problems. Don't reinvent the wheel!



# Understanding Conditional Statements and Loops in Python

This presentation provides an overview of conditional statements and loops in Python programming. Learn how to use them effectively with the help of examples.

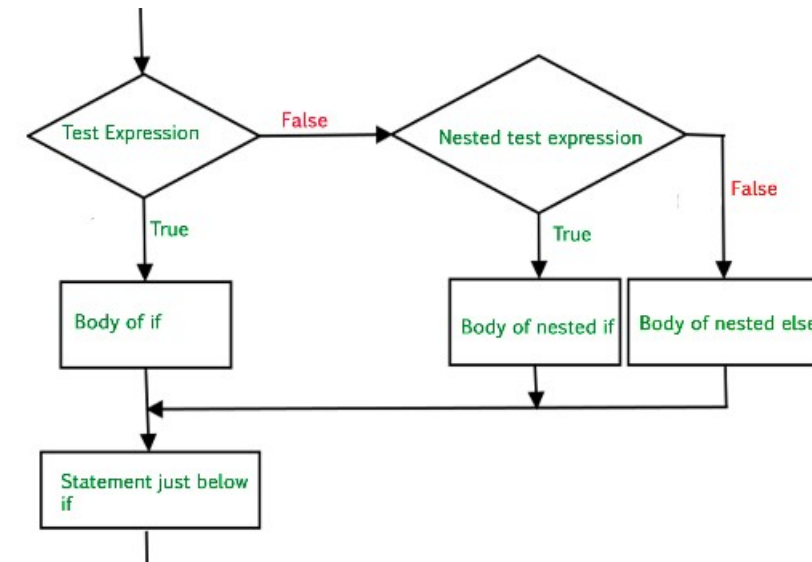
# What are Conditional Statements in Python?

In programming, conditional statements are used to make decisions. In Python, it helps to execute certain code only if a condition is satisfied. Explore the syntax and some basic examples of conditional statements in this section.

```
[2]: a = 10
      b = 10
      if a == b:
          → print('yes')
      else:
          → print('no')
```

## If-else Statements

The if-else statement is used to execute different blocks of code depending upon the condition. It is often used in decision making and validation.

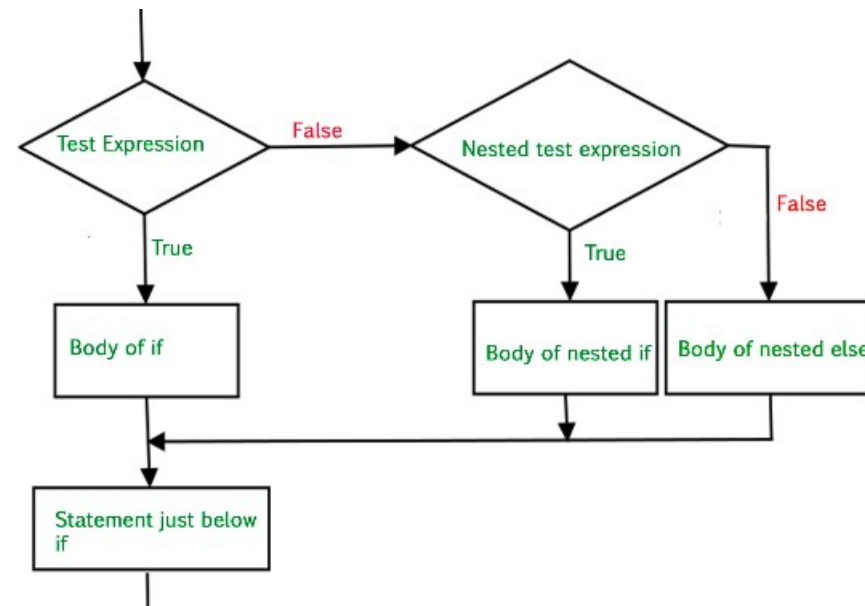


## Nested If Statements

The nested if statement is used to execute a condition within a condition. It is used when we want to check multiple conditions one after the other.

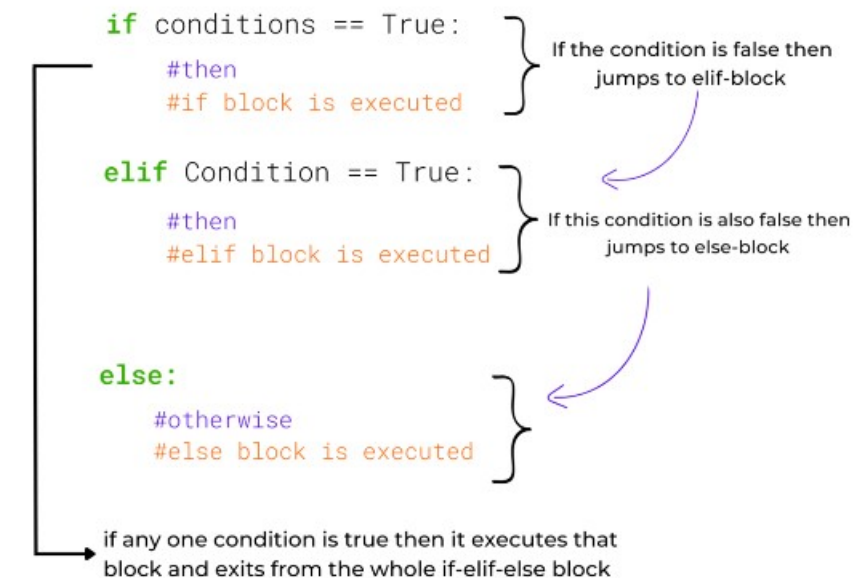
# Nested If Statements in Python

Nested if statements can be used to test conditions within conditions. Discover how to use nested if statements effectively with the help of some basic examples.



## Nested If-else Statements

You can use multiple if-else statements within each other to test multiple conditions. For example, if  $(x > y)$ : if  $(y > z)$ : print("y is greater than z").



## If-elif-else Statements

You can use if-elif-else statements when you have multiple conditions to test. elif is short for else if. For example, if  $(x > y)$ : print("x is greater than y") elif  $(y > x)$ : print("y is greater than x") else: print("x and y are equal")

# What are Loops in Python?

The loops in Python help to run a block of code repeatedly until the given condition is satisfied. Explore the syntax and some basic examples of loops in this section.

## For Loops

The for loop in Python is used to iterate over a sequence. It can iterate over a range of numbers, list, tuples, and more. We can use break and continue statements to control the flow of the loop.

## While Loops

The while loop in Python repeats a block of code as long as a given condition is true. It is used when the number of iterations can't be decided before the loop starts. We can use break and continue statements to control the flow of the loop.



# For Loops in Python

The for loop in Python is used to iterate over a sequence. Learn how to use the for loop effectively by following some basic examples.

**1**

## Range Function

You can use the range function to iterate over a fixed set of numbers. For example,

```
for i in range(5):  
    print(i)
```

**2**

## Iterating Over Lists

You can use a for loop to iterate over a list.

For example,

```
num_list=[1, 2, 3]  
for num in num_list:  
    print(num)
```

# While Loops in Python

The while loop in Python is used to repeat a block of code as long as a given condition is true. Learn how to use the while loop effectively with the help of some basic examples.

| Loop Type                 | Description   | Example  |
|---------------------------|---|--|
| Infinite Loop             | The loop keeps executing indefinitely until the given condition becomes false.                      | <code>while True: print("Hello World")</code>                            |
| Loop with Counter         | The loop keeps executing until the counter reaches a given value.                                   | <code>i = 1; while i &lt;= 5: print(i); i += 1</code>                    |
| Loop with Break Condition | The loop keeps executing until a given condition becomes false or a break statement is encountered. | <code>i = 1; while i &lt;= 10: if i == 5: break; print(i); i += 1</code> |



# Conclusion

In conclusion, conditional statements and loops are powerful tools in Python that help to make decisions and execute code repeatedly. By understanding how to use them effectively, you can become a better programmer. Remember to practice and experiment with different examples to reinforce your learning.

# List operations in Python

In this presentation, we will explore the fundamentals of Python list operations, including creating, modifying, and looping through lists.



# What are Python lists?



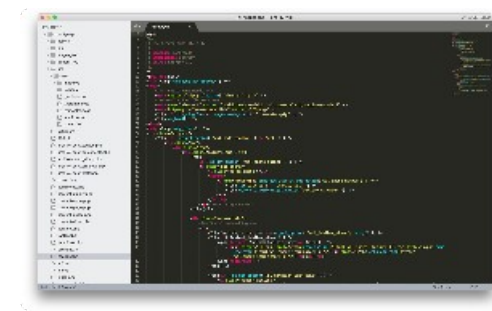
## Python

Python is a high-level programming language used in web development, data science, and machine learning.



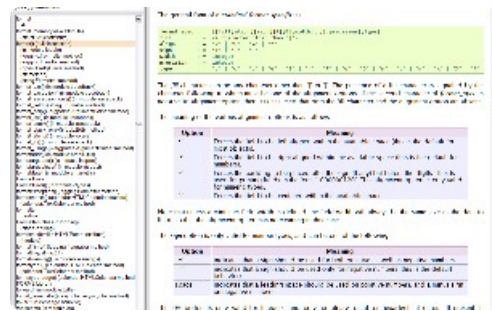
## Lists

Python Lists are ordered, mutable data structures containing items of multiple data types.



## Code Editors

Popular code editors for Python include IDLE, PyCharm, VSCode, and Sublime Text.



## Documentation

Python documentation is available online and provides comprehensive information about list operations and other functionalities.

# Creating lists

## Initialization

Lists can be initialized with a set of values using square brackets.

## Concatenation

Lists can be concatenated using the '+' operator.

## List Comprehension

List comprehension allows for the creation of lists using a single line of code.

# Indexing and Slicing

1

## Indexing

Lists can be accessed using their index numbers. Negative indices are used to access elements from the end.

2

## Slicing

Lists can be sliced to create a new list containing a subset of the original.

3

## Extended slicing

Extended slicing allows for more flexibility in slicing operations, using a start-stop-step notation.



# Modifying lists



## Appending

New items can be added to a list using the `append()` method.



## Inserting

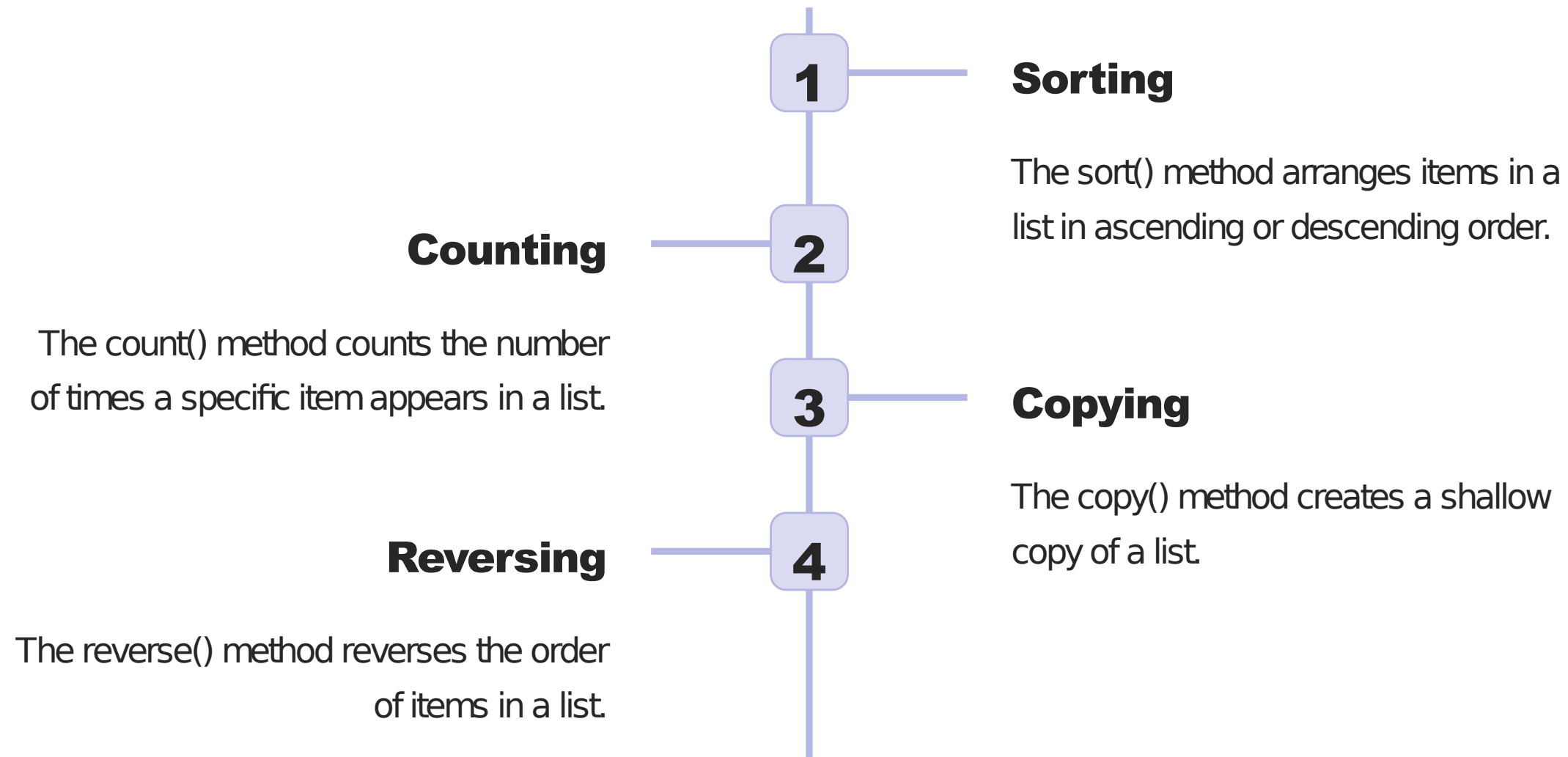
New items can be inserted into a specific position using the `insert()` method.



## Deleting

Items can be removed from a list using the `remove()` and `del` methods, or by slicing.

# List methods



# Looping through lists

## **For Loop**

For loops allow for the iteration through each item in a list and the execution of a block of code.

## **While Loop**

While loops allow for the iteration through a list until a specific condition is met.

## **List comprehension**

List comprehension can be used to simplify the process of looping through lists.

# List comprehension



## Syntax

List comprehension uses a compact syntax to create a new list from an existing list.



## Expression

The expression specifies the rules for transforming the original list into the new list.



## Conditionals

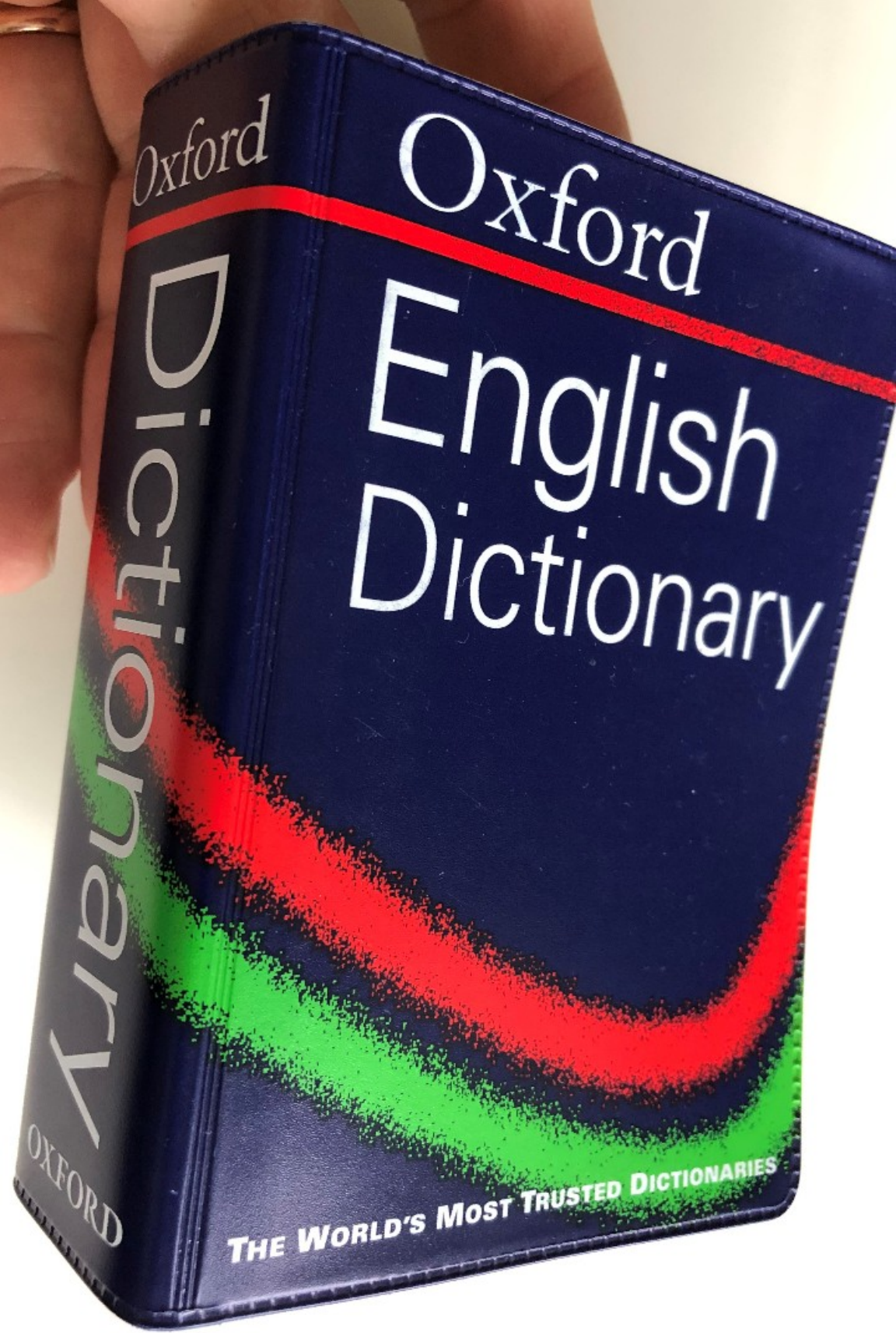
Conditionals can be added to filter data from the original list before creating a new list.

# In Conclusion

Python list operations are an essential part of the language and allow for the efficient manipulation of data structures. Regular use of this functionality can greatly improve coding productivity and efficiency.







# Dictionary Operations in Python

Welcome to our guide on dictionary operations in Python! Learn how to create, access, modify, and iterate over dictionaries in Python.

# Creating Dictionaries

Python provides a simple syntax for creating dictionaries. Add and update key-value pairs to build your dictionary.

**1**

## Syntax

Use curly braces and colons to define key-value pairs.

**2**

## Adding & Updating

Use assignment to add or update key-value pairs.



# Creating a Dictionary from..

## Assigning Values

Dictionaries store key-value pairs. Use curly braces to enclose the values, with each key-value pair separated by a colon.

## From Sequences

You can also use lists or tuples to create a dictionary using the `dict()` method, where each element of the sequence represents a key-value pair.

## From Variables

Creating a new dictionary and adding items to it using variables is an effective way of creating a dictionary with minimal code.

# Dictionary Methods and Properties

## **keys()**

Returns a list of all the keys in the dictionary.

## **values()**

Returns a list of all the values in the dictionary.

## **items()**

Returns a list of key-value pairs in the dictionary.

## **len()**

Returns the number of items in the dictionary.

# Examples of Dictionary Operations in Python

## Scrabble Word Scores

Python dictionary can be used to quickly calculate the score of a word in Scrabble. Each letter is assigned a value, which can be stored in a dictionary.

## Phonebook

A popular use case for dictionaries is to store a phonebook. Each entry can be stored as a key-value pair, where the key is the name and the value is the phone number.

## Weather App

An application that gets the weather forecast from an API and displays it to the user. The API response can be stored in a dictionary to make it easier to parse and display the data.

# Accessing Values in a Dictionary

## 1 Keys

Dictionary keys can be accessed using square brackets or the `get()` method. If the key does not exist, the `get()` method can provide a default value instead of returning `None`.

## 2 Values

Dictionary values are accessed using the variable name followed by the key in square brackets or `values()` method.

## 3 Items

You can access items (both key-value pairs) in a dictionary using `items()` method.

# Dictionary Methods

There are several built-in methods in Python to perform common tasks with dictionaries.

| Dictionary Methods          |   |
|-----------------------------|---|
| Nonmutating Methods         |   |
| <code>D.copy()</code>       | Returns a shallow copy of the dictionary (a copy whose items are the same objects as <i>D</i> 's, not copies thereof)               |
| <code>D.has_key(k)</code>   | Returns <code>True</code> if <i>k</i> is a key in <i>D</i> ; otherwise, returns <code>False</code> , just like <i>k</i> in <i>D</i> |
| <code>D.items()</code>      | Returns a new list with all items (key/value pairs) in <i>D</i>   |
| <code>D.keys()</code>       | Returns a new list with all keys in <i>D</i>  |
| <code>D.values()</code>     | Returns a new list with all values in <i>D</i>  |
| <code>D.iteritems()</code>  | Returns an iterator on all items (key/value pairs) in <i>D</i>  |
| <code>D.iterkeys()</code>   | Returns an iterator on all keys in <i>D</i>   |
| <code>D.itervalues()</code> | Returns an iterator on all values in <i>D</i>   |
| <code>D.get(k[, x])</code>  | Returns <i>D</i> [ <i>k</i> ] if <i>k</i> is a key in <i>D</i> ; otherwise, returns <i>x</i> (or <code>None</code> , if <i>x</i> is |

## Examples

Explore examples of dictionary methods, such as `keys()`, `values()`, and `items()`.

```
first_tiny_script.py — ~/Documents/python/automate
first_tiny_script.py
1  # This program says hello and asks for my name
2
3  print('Hello world!')
4  print('What is your name?') # ask for their name
5  myName = input()
6  print('It is good to meet you, ' + myName)
7  print('The length of your name is:')
8  print(len(myName))
9  print('What is your age?') # ask for their age
10 myAge = input()
11 print('You will be ' + str(int(myAge) + 1) + ' in a year.')
12
```

## Common Methods

Discover frequently used methods like `get()`, `update()`, and `pop()` for manipulating dictionaries.

# Adding or Changing Items in a Dictionary

```
class BigFile:
    def __init__(self, datadir, ndims):
        idfile = os.path.join(datadir, "id.txt")
        self.names = [x.strip() for x in str.split(open(idfile).read()) if x.strip()]
        self.name2index = dict(zip(self.names, range(len(self.names))))
        self.ndims = ndims
        self.featurefile = os.path.join(datadir, "feature.bin")
        print "[BigFile] %d features, %d dimensions" % (len(self.names), self.ndims)
        print "    binary: %s" % self.featurefile
        print "    txt: %s" % idfile
        print " "
    def read(self, requested, isname=True):
        if isname:
            index_name_array = [(self.name2index[x], x) for x in requested if x in self.names]
            assert(min(requested) >= 0)
            assert(max(requested) < len(self.names))
            index_name_array = [(x, self.names[x]) for x in requested]
            index_name_array.sort()
            index_name_array.sort()
            vecs = seq_read(self.featurefile, self.ndims, [x[0] for x in index_name_array])
            return [x[1] for x in index_name_array], vecs
        else:
            return (self.names, self.ndims)
```

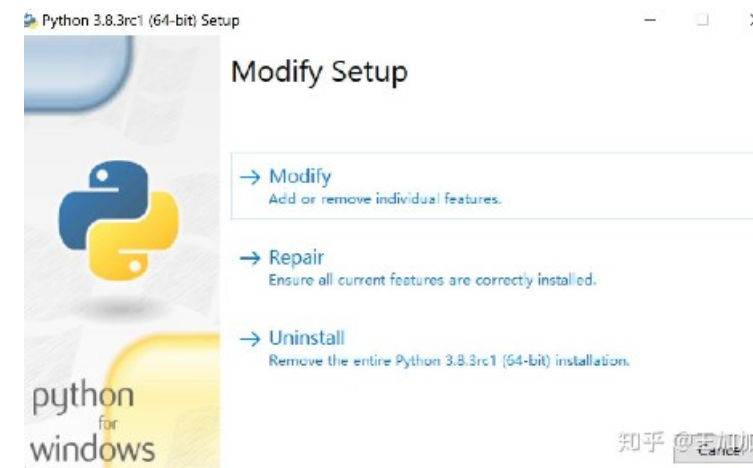
## Add Key-Value Pairs

To add a new item to a dictionary, use square brackets with a new key and its corresponding value.

```
class BigFile:
    def __init__(self, datadir, ndims):
        idfile = os.path.join(datadir, "id.txt")
        self.names = [x.strip() for x in str.split(open(idfile).read()) if x.strip()]
        self.name2index = dict(zip(self.names, range(len(self.names))))
        self.ndims = ndims
        self.featurefile = os.path.join(datadir, "feature.bin")
        print "[BigFile] %d features, %d dimensions" % (len(self.names), self.ndims)
        print "    binary: %s" % self.featurefile
        print "    txt: %s" % idfile
        print " "
    def read(self, requested, isname=True):
        if isname:
            index_name_array = [(self.name2index[x], x) for x in requested if x in self.names]
            assert(min(requested) >= 0)
            assert(max(requested) < len(self.names))
            index_name_array = [(x, self.names[x]) for x in requested]
            index_name_array.sort()
            index_name_array.sort()
            vecs = seq_read(self.featurefile, self.ndims, [x[0] for x in index_name_array])
            return [x[1] for x in index_name_array], vecs
        else:
            return (self.names, self.ndims)
```

## Update the Dictionary

A dictionary can be updated using the update()



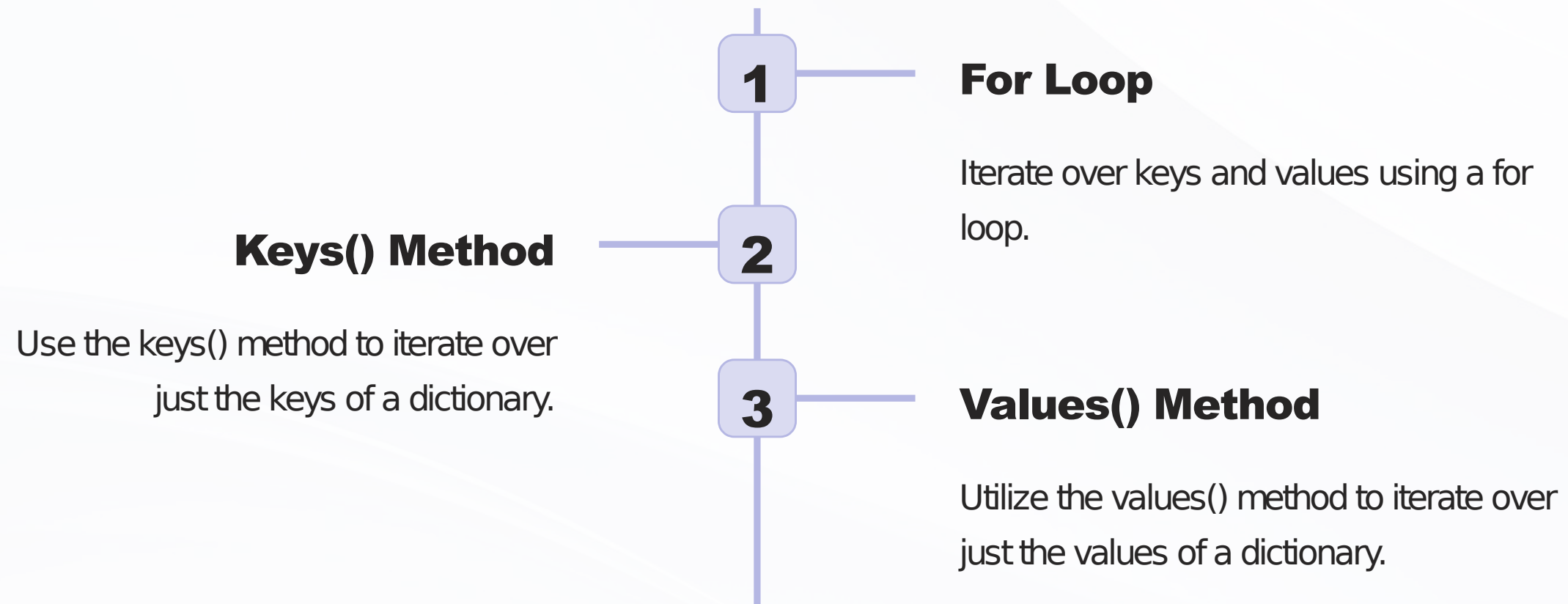
## Modify a Value

Updating the value of an existing item is done using square brackets with the key and assigning a new value.

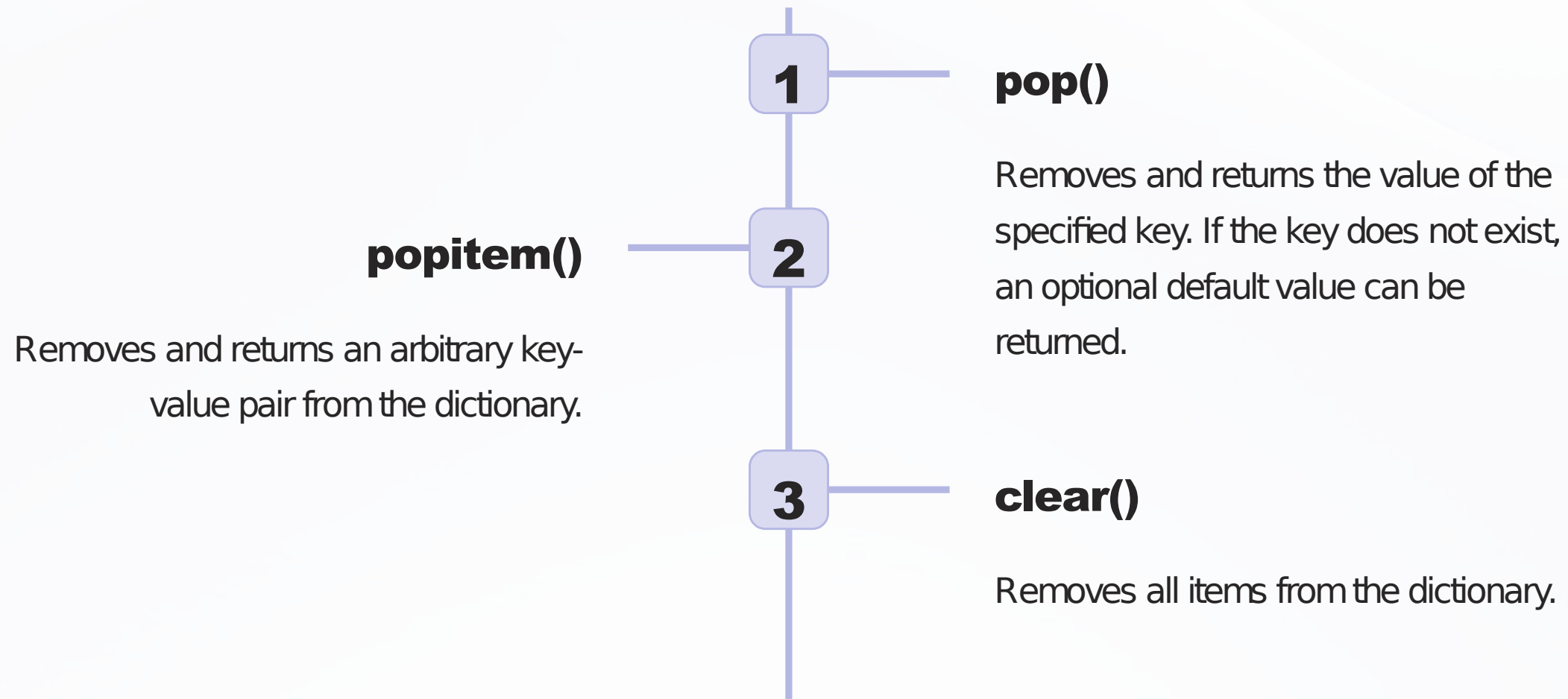


# Iterating Over Dictionary Elements

Learn different ways to iterate through the elements of a dictionary and perform operations on them.



# Removing Items from a Dictionary



# Wrap Up



## You did it!

Congratulations on completing our guide to dictionary operations in Python. You should now have a solid understanding of how to create, access, modify, and remove items from a dictionary.

```
first_tiny_script.py — ~/Documents/python/automate
first_tiny_script.py
1  # This program says hello and asks for my name
2
3  print('Hello world!')
4  print('What is your name?') # ask for their name
5  myName = input()
6  print('It is good to meet you, ' + myName)
7  print('The length of your name is:')
8  print(len(myName))
9  print('What is your age?') # ask for their age
10 myAge = input()
11 print('You will be ' + str(int(myAge) + 1) + ' in a year.')
12
```

## Practice Makes Perfect

The best way to become proficient in using dictionaries in Python is to practice. Try building your own projects or working through coding challenges that utilize dictionaries.

# WHAT IS A PYTHON TUPLE

## Tuple Operations in Python

Learn how to effectively work with tuples in Python to streamline your programming.



# Tuple Basics

```
>>> a = [1, 2, 3]
>>> a[0] = 100
>>> a
[100, 2, 3]
>>> b = (1, 2, 3)
>>> b[0]
1
>>> b[1:3]
(2, 3)
>>> len(b)
3
>>> print(b)
```

- Any operation that you can do on lists, and that does **NOT** change contents, you can do on tuples.
- Examples:
  - Indexing, e.g., b[0]
  - Slicing, e.g., b[1:3]
  - Taking the length, e.g., as len(b)
  - Printing, e.g., print(b)
- Operations that change contents of lists, produce errors on tuples.
- Examples: list methods pop, insert, reverse, sort, append

```
console
sole 1/A ✖

runfile('/home/linuxhint/Documents/numbers.py', wdir='/home/linuxhint')

3, 4, 5, 6, 7, 8, 9)
.2, 1.3, 1.4, 1.5)
', 'welcome', 'to', 'linuxhint')
, 'HELLO', (2+3j))
'int', 'linuxhint', 'linuxhint', 'linuxhint', 'linuxhint')
[1, 2, 3])
3, 10.3, 'kamran')
'int',)
```

```
coord = 1, 2
(1, 2)
coord
1
coord = (1, 2)
(1, 2)
coord
(1, 2)

Type 'help', 'copyright', 'credits' or 'license()'
>>> a, b = 1, 2
>>> a, b := 1, 2
(1, 1, 2)
>>> a, b
(1, 1)

>>> (x, y) = 1, 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot unpack non-iterable int object
>>> (x, y) := (1, 2)
(1, 2)
>>> x, y
(1, 2)

>>> def f():
...     return (1, 2)
...
>>> a, b := f()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'e' is not defined

>>> a, b = 1, 2
>>> a, b := f()
(1, (1, 2))
>>> a, b
(1, (1, 2))
```

## Definition

Tuples are an ordered, immutable sequence of elements. They use parentheses instead of square brackets, like lists.

## Creating tuples

You can create a tuple by separating elements with commas or by using the tuple() method.

## Tuple unpacking

You can easily assign values to multiple variables in a single line of code using tuple unpacking. This feature is unique to tuples.

# Accessing Individual Elements

## Indexing

You can access tuple elements using their index number, starting from 0.

## Slicing

You can access a subset of elements by slicing the tuple. Slicing is done using the syntax [start:end:step].

## Negative indexing

You can also index from the end of the tuple using negative numbers, starting from -1.

## Unpacking

You can easily assign values to multiple variables in a single line of code using tuple unpacking. This feature is unique to tuples.



# Modifying Elements

1

## Immutable

Tuples are immutable, meaning you cannot modify individual elements once the tuple is created.

2

## Workaround

The only way to modify a tuple is to create a new tuple with the modified value.

3

## Advantages

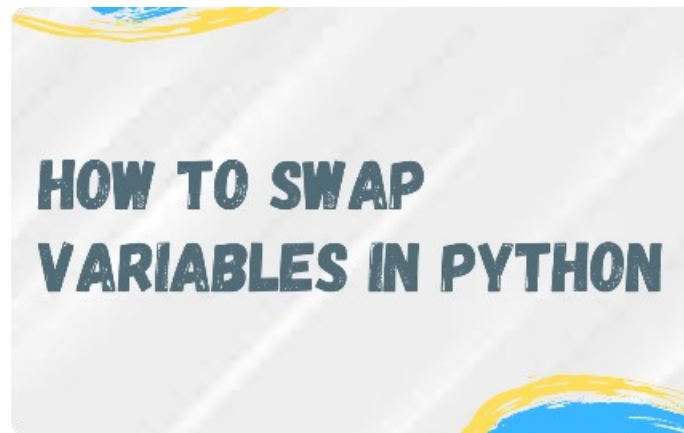
This makes tuples ideal for storing data that should not be changed, such as configuration settings.

# Swapping Values



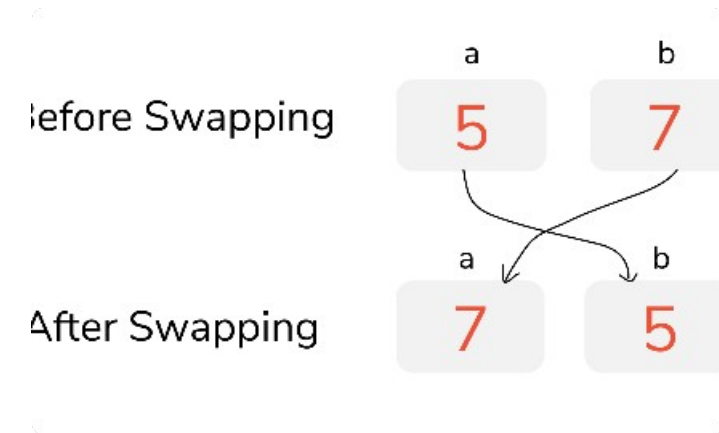
## Single line of code

You can easily swap the values of two variables using tuples with a single line of code.



## Benefits

Tuple swapping is a simple and elegant way to exchange values without having to use a temporary variable.



## Applications

It's commonly used for sorting algorithms or for switching the values of variables within loops.

# Concatenating Tuples

## 1 Definition

You can combine two or more tuples into a single tuple using the `+` operator.

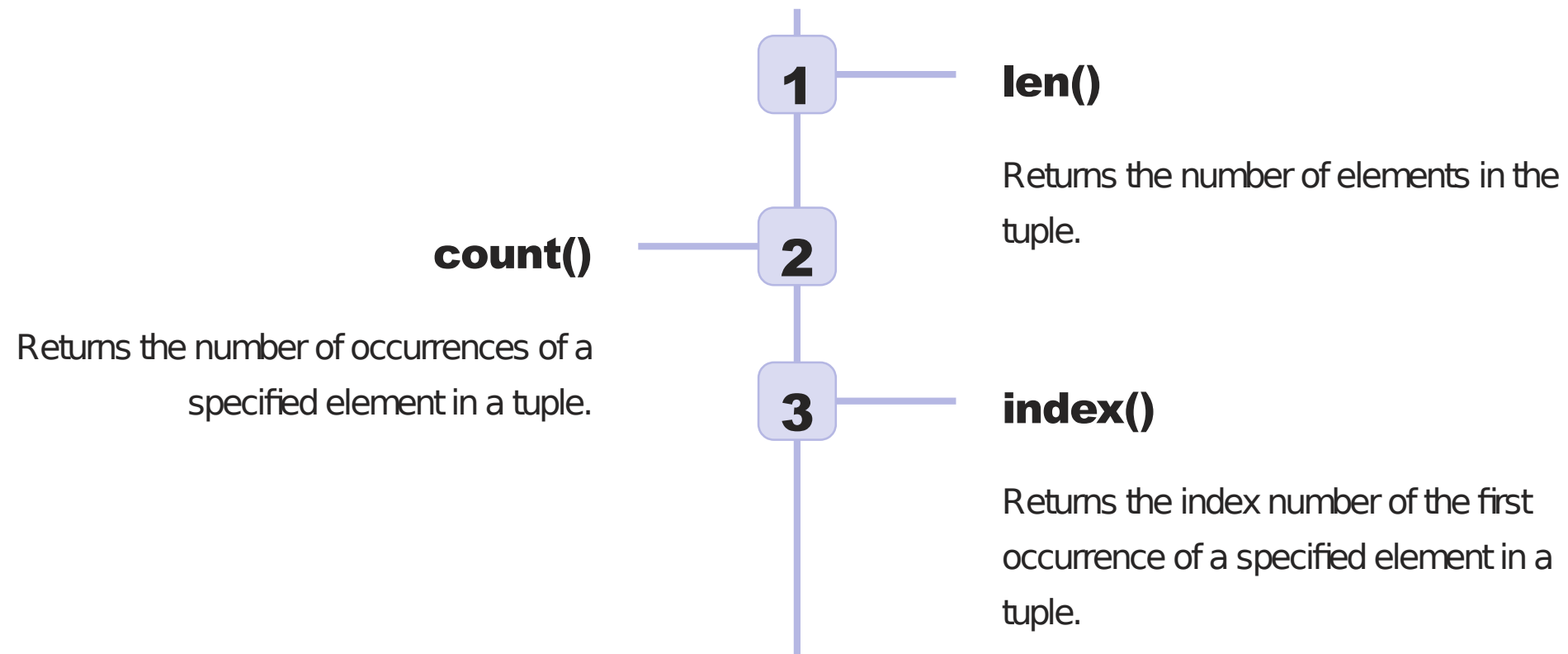
## 2 Benefits

Concatenating tuples allows you to combine data from multiple sources into a single object.

## 3 Limitations

Since tuples are immutable, concatenation creates a new tuple and can be costly for large tuples.

# Built-in Tuple Methods



# Conclusion



## Practice makes perfect

By mastering tuple operations, you'll be more effective in your Python programming and be able to create efficient and readable code.



## Talk less, code more!

Dive into these concepts, start practising and contribute to the Python community!



## Cheers to a brighter tomorrow!

Remember to enjoy the journey.





# Set Operations in Python

Learn about the powerful set operations in Python and how they can simplify your code. Discover how to combine, compare, and manipulate sets effortlessly.



# Union

## Example

$\text{nums} = \{1, 2, 3\} \cup \{3, 4, 5\} \rightarrow \{1, 2, 3, 4, 5\}$

1

## Set Combination

Combine two or more sets into one, eliminating duplicates and preserving unique elements.

2

3

## Use Case

Merging data from multiple sources to create a unified dataset.

# Intersection

## Finding Common Elements

Identify elements that are present in multiple sets.

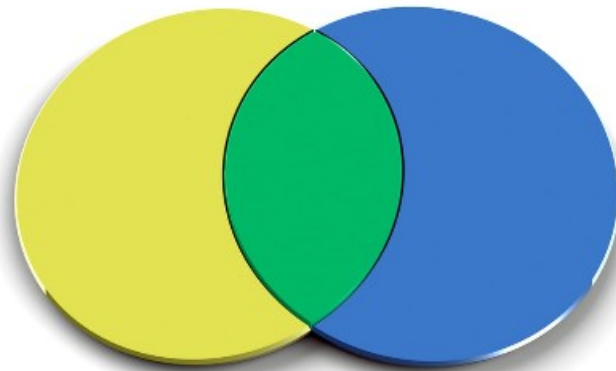
## Example

fruits = {'apple', 'banana',  
'orange'} ∩ {'banana', 'kiwi'}  
→ {'banana'}

## Use Case

Checking if two sets of users have any mutual connections.

# Difference



## Finding Unique Elements Example

Determine elements that exist in one set but not in another.

`odd_nums = {1, 3, 5, 7, 9} - {3, 7} → {1, 5, 9}`

## Use Case

Removing unwanted items from a dataset

# Subset

## Example

colors = {'red', 'green', 'blue'}  $\subseteq$  {'red', 'green', 'blue', 'yellow'}  $\rightarrow$  True

1

## Determine Inclusion

Check if one set is entirely contained within another set.

2

3

## Use Case

Verifying the presence of a specific criteria within a larger collection of data.

# Superset

## 1 Check Containment

Determine if a set contains all the elements of another set.

## 2 Example

`numbers = {1, 2, 3, 4} ⊇ {1, 3} → True`

## 3 Use Case

Ensuring that a dataset covers all the necessary data points for analysis.

# Symmetric Difference

## Finding Exclusive Elements

Identify elements that are present in either of the sets, but not in both.

## Example

$\text{set1} = \{4, 5, 6\} \ominus \{5, 6, 7\} \rightarrow \{4, 7\}$

## Use Case

Comparing two lists of features to identify those that exist exclusively in one list or the other.



# Conclusion

Mastering set operations in Python will empower you to manipulate data efficiently and effectively. Start harnessing the full potential of sets today!



# Exploring Function, Modules and Packages in Python

Python has a wealth of useful functions, modules, and packages that can make programming a lot easier. In this presentation, we will go over some basics and examples that will help you master the art of Python programming!



python

```
6 from watson.events import types
7 from watson.framework import events
8 from watson.http.messages import Response, Request
9 from watson.common.imports import get_qualified_name
10 from watson.common.contextmanagers import suppress_logging
11
12
13 ACCEPTABLE_RETURN_TYPES = (str, int, float, bool)
14
15
16 class Base(ContainerAware, metaclass=abc.ABCMeta):
17     """The base class for all controllers.
18     Attributes:
19         __action__ (string): The last action performed.
20     """
21
22     def execute(self, **kwargs):
23         method = self.get_execute_method(**kwargs)
24         self.__action__ = method.__name__
25         return method(**kwargs) or {}
26
27     @abc.abstractmethod
28     def get_execute_method(self, **kwargs):
29         raise NotImplementedError(
30             "You must implement get_execute_method"
31         )
```

# Functions in Python

Functions allow us to write reusable code, while keeping our code organized and easy to understand. They can also return values and accept parameters. Here's an example:

```
def square(a):  
    """  
    Square the input add add 1  
    """  
    c=1  
    b=a*a+c;  
    print(a, " if you square+1 ",b)  
    return(b)  
  
x=2;  
z= square(x)
```

Annotations: `a` is a Formal parameter; `c=1` is a Local variable. Brackets indicate the Function definition and Main program code.

```
def square(a):  
    """  
    Square the input add add 1  
    """  
    c=1  
    b=a*a+c;  
    print(a, " if you square+1 ",b)  
    return(b)  
  
x=2;  
z= square(x)
```

Annotations: `a` is a Formal parameter; `c=1` is a Local variable. Brackets indicate the Function definition and Main program code.

```
def square(a):  
    """  
    Square the input add add 1  
    """  
    c=1  
    b=a*a+c;  
    print(a, " if you square+1 ",b)  
    return(b)  
  
x=2;  
z= square(x)
```

Annotations: `a` is a Formal parameter; `c=1` is a Local variable. Brackets indicate the Function definition and Main program code.

## Syntax

A function in Python begins with the keyword 'def', followed by the function name and a set of parentheses.

## Example

Here's a simple function that takes in a name and returns a personalized greeting:

```
def greeting(name):  
    return "Hello, " + name
```

## Structure

The body of the function must be indented, typically by 4 spaces. A function can also include comments that describe its use.

# Modules in Python

A module is a file containing Python definitions and statements. They can be used to organize code and make it easier to read. Here's more information:

1

## What is a module?

A module is a Python file containing code for specific functionality.

2

## Types of modules

There are built-in modules that come with Python and third-party modules that can be downloaded.

3

## Importing modules

Use the 'import' statement followed by the module name to use a module in your code.

4

## Example of using a module

Here is an example of using the 'math' module:

```
import math  
print(math.sqrt(25))
```

# Packages in Python

Packages are collections of modules. They can be used to organize code into a hierarchical structure, making it even easier to read and understand. Here are some things you should know:

## Understanding packages

A package is a folder that contains one or more modules or sub-packages.

## Creating and organizing packages

To create a package, you need to create a folder with an `'__init__.py'` file in it. You can then organize modules and sub-packages within this folder.

## Using packages in a project

You can then use the `'import'` statement to import modules or sub-packages from your package folder.

## Example of using a package

Here is an example of a package called `'my_package'`, containing the `'my_module'` module:

```
import my_package.my_module
```





# Conclusion

You've learned about the basics of functions, modules and packages in Python. You can now create your own functions, import modules, and use packages to make your code more organized. Keep practicing and exploring more modules and packages as you become a Python expert!