Introduction to AI

What is Artificial Intelligence (AI)?

Artificial intelligence is a field of science concerned with building computers and machines that can reason, learn, and act in such a way that would normally require human intelligence or that involves data whose scale exceeds what humans can analyze.

AI is a broad field that encompasses many different disciplines, including computer science, data analytics and statistics, hardware and software engineering, linguistics, neuroscience, and even philosophy and psychology.

On an operational level for business use, AI is a set of technologies that are based primarily on machine learning and deep learning, used for data analytics, predictions and forecasting, object categorization, natural language processing, recommendations, intelligent data retrieval, and more.

https://cloud.google.com/learn/what-is-artificial-intelligence

History of AI :

Pre-20th Century:

- **1726:** Jonathan Swift's "Gulliver's Travels" introduces "The Engine," a fictional device that assists scholars in generating new ideas, anticipating concepts of algorithmic text generation.

Early 20th Century:

- **1914:** Spanish engineer Leonardo Torres y Quevedo demonstrates "El Ajedrecista," an automated chess-playing machine capable of endgame scenarios without human intervention.
- **1921:** Karel Čapek's play "Rossum's Universal Robots" (R.U.R.) introduces the term "robot," derived from the Czech word "robota," meaning forced labor.

Mid-20th Century:

- **1939:** John Atanasoff and Clifford Berry develop the Atanasoff-Berry Computer (ABC), one of the earliest digital electronic computers, introducing binary computation and electronic circuits.

- **1943:** First Artificial Neurons : Warren McCulloch and Walter Pitts designed the first artificial neurons, laying the groundwork for neural networks. This work introduced a mathematical model for neural activity, which was pivotal in understanding how machines could mimic human brain functions.
- **1950:** Alan Turing publishes "Computing Machinery and Intelligence," proposing the Turing Test to assess a machine's ability to exhibit intelligent behavior indistinguishable from humans.
- **1956:** The Dartmouth Conference, organized by John McCarthy and others, marks the official birth of AI as an academic discipline, coining the term "artificial intelligence."

Late 20th Century:

- **1960s-1970s:** AI research focuses on problem-solving and symbolic methods, but limitations lead to the first "AI winter," a period of reduced funding and interest.
- **1980s:** The rise of expert systems, which emulate decision-making abilities of human experts, revitalizes AI research, though eventual limitations lead to another decline.

Early 21st Century:

- **1990s-2000s:** Advancements in machine learning and increased computational power reignite AI research, leading to applications in data analysis and pattern recognition.
- **2010s:** The emergence of deep learning and neural networks results in significant breakthroughs in image and speech recognition, natural language processing, and autonomous systems.

Recent Developments:

- **2020s:** Models like OpenAI's Generative Pretrained Transformer (GPT) series gain widespread attention for their ability to generate human-like text, marking significant steps toward realizing the ancient dream of intelligent machines.

https://www.ibm.com/think/topics/history-of-artificial-intelligence

Different Fields of AI :

- **Machine Learning (ML)**
  Machine Learning is a subset of AI that focuses on developing algorithms that enable computers to learn from and make predictions based on data. It encompasses various techniques, including supervised learning, unsupervised learning, and reinforcement learning. ML is widely used in applications such as image recognition, recommendation systems, and predictive analytics.

- **Deep Learning**
  Deep Learning is a specialized area within Machine Learning that employs artificial neural networks with multiple layers (deep neural networks) to model complex patterns in large datasets. This approach has led to breakthroughs in various domains such as computer vision and natural language processing due to its ability to automatically extract high-level features from raw data.

- **Natural Language Processing (NLP)**
  NLP is the field focused on the interaction between computers and human language. It enables machines to understand, interpret, generate, and respond to human language in a meaningful way. Applications of NLP include language translation, sentiment analysis, chatbots, and text summarization. Techniques used in NLP often involve machine learning and linguistic rules to handle tasks effectively.

- **Computer Vision**
  Computer Vision aims to enable machines to interpret and understand visual information from the world. This field involves developing algorithms that can perform tasks such as image classification, object detection, facial recognition, and scene understanding. Techniques like convolutional neural networks (CNNs) are commonly used in this domain to analyze visual data.

- **Robotics**
  Robotics integrates AI with mechanical engineering to create robots capable of performing tasks autonomously or semi-autonomously. This field encompasses various applications, from industrial automation to personal assistants. Robotics relies on AI for navigation, manipulation, and interaction with the environment.

- **Speech Recognition**
  Speech Recognition is a technology that enables machines to understand and process human speech. It involves converting spoken language into text and is essential for applications like virtual assistants (e.g., Siri or Alexa), transcription services, and voice-controlled systems. This field often intersects with NLP for better context understanding.

- **Expert Systems**
  Expert Systems are AI programs that emulate the decision-making abilities of a human expert in specific domains. They use a knowledge base combined with a set of rules to provide recommendations or solutions for complex problems. Applications include medical diagnosis systems and financial forecasting tools.

- **Fuzzy Logic**
  Fuzzy Logic deals with reasoning that is approximate rather than fixed or exact. This approach allows AI systems to handle uncertainty and vagueness in data. It is applied in control systems, decision-making processes, and various engineering applications where binary true/false logic may not be sufficient.

- **Evolutionary Computation**
  Evolutionary Computation involves algorithms inspired by the process of natural selection. These algorithms are used for optimization problems where traditional methods may struggle. Techniques include genetic algorithms which evolve solutions over generations based on fitness criteria.

- **Swarm Intelligence**
  Swarm Intelligence is based on the collective behavior of decentralized systems, often inspired by natural phenomena such as bird flocking or fish schooling. This field focuses on optimizing problem-solving through cooperation among agents or entities.

AI applications in various industries & Real-world use cases :

- **Healthcare**
  AI is revolutionizing healthcare by enabling personalized medicine, improving diagnostic accuracy, and streamlining administrative processes. For instance, AI algorithms analyze medical images to detect diseases early, while chatbots assist patients in scheduling appointments and managing inquiries.

- **Finance**
  In the finance sector, AI enhances security through fraud detection systems that analyze transaction patterns to identify suspicious activities. Additionally, algorithmic trading utilizes AI to execute trades at optimal times based on market data analysis, significantly improving investment strategies.

- **Retail**
  Retailers leverage AI for personalized shopping experiences by analyzing customer behavior to offer tailored product recommendations. Smart inventory management systems utilize predictive analytics to optimize stock levels and reduce waste, enhancing overall operational efficiency.

- **Manufacturing**
  AI applications in manufacturing include predictive maintenance, where machines are monitored using sensors to predict failures before they occur. This approach minimizes downtime and extends equipment lifespan. AI also enhances quality control through computer vision systems that detect defects in products during production.

- **Logistics and Transportation**
  AI optimizes logistics through route optimization algorithms that analyze traffic patterns and delivery schedules to reduce fuel consumption and improve delivery times. Companies like FedEx employ AI for smart package sorting and real-time tracking, enhancing operational efficiency.

- **Energy**
  The energy sector utilizes AI for predictive maintenance of equipment and optimizing energy production through demand forecasting. For example, ExxonMobil uses AI for reservoir modeling to improve oil recovery rates and manage resources more effectively.

- **Telecommunications**
  Telecommunications companies implement AI for network optimization and predictive maintenance. By analyzing network traffic data, AI can identify potential issues before they disrupt service, ensuring a seamless experience for users.

- **Agriculture**
  In agriculture, AI technologies are employed for precision farming, where data from sensors and drones is analyzed to monitor crop health and optimize resource use. This leads to improved yields and sustainable farming practices.

- **Automotive**
  The automotive industry employs AI in the development of autonomous vehicles, utilizing machine learning algorithms to process data from sensors for navigation and safety. Additionally, manufacturers use AI for optimizing production processes.

- **Hospitality**
  In hospitality, AI enhances guest experiences through personalized services offered by chatbots and virtual assistants. These systems can manage bookings, provide recommendations, and respond to customer inquiries efficiently.

Real-World Use Cases

- **Healthcare Example:** American Addiction Centers reduced employee onboarding time from three days to just 12 hours using AI tools that streamline administrative tasks, thus improving patient care efficiency.
- **Finance Example:** Scotiabank utilizes Gemini and Vertex AI to create a personalized banking experience through an award-winning chatbot that assists clients with their banking needs.
- **Retail Example:** Netflix employs machine learning algorithms to recommend shows based on user viewing history, significantly enhancing user engagement and satisfaction.
- **Manufacturing Example:** AES uses generative AI agents for energy safety audits, resulting in a drastic reduction in audit costs from 14 days to just one hour while increasing accuracy by 10-20%.
- **Logistics Example:** FedEx implements AI-driven smart package sorting systems that automate the sorting process in distribution centers, leading to improved handling efficiency.
- **Energy Example:** Copel transformed data access with Google Cloud AI to enable employees to extract real-time insights from their ERP system using natural language queries.

Common Terminologies in AI:

- **Artificial Intelligence (AI)**
  The simulation of human intelligence processes by machines, particularly computer systems. AI encompasses various subfields, including machine learning, natural language processing, and robotics.

- **Machine Learning (ML)**
  A subset of AI that enables systems to learn from data and improve their performance over time without being explicitly programmed. ML algorithms identify patterns in data and make predictions or decisions based on those patterns.

- **Deep Learning**
  A specialized form of machine learning that uses neural networks with multiple layers to analyze various factors of data. Deep learning is particularly effective for tasks such as image and speech recognition.

- **Natural Language Processing (NLP)**
  The ability of machines to understand, interpret, and generate human language. NLP combines linguistics and AI to enable computers to process and analyze large amounts of natural language data.

- **Generative Adversarial Network (GAN)**
  A framework for training models using two neural networks: a generator that creates data and a discriminator that evaluates it. GANs are widely used in generating realistic images, videos, and other media.

- **Algorithm**
  A step-by-step procedure or formula for solving a problem. In AI, algorithms are used to process data and make decisions based on specific criteria.

- **Neural Network**
  A computational model inspired by the human brain, consisting of interconnected nodes (neurons) that process information. Neural networks are fundamental to deep learning applications.

- **Supervised Learning**
  A type of machine learning where models are trained on labeled datasets, meaning that the input data is paired with the correct output. This approach is commonly used for classification and regression tasks.

- **Unsupervised Learning**
  A machine learning technique where models are trained on data without labeled responses. The goal is to identify patterns or groupings within the data.

- **Reinforcement Learning**
  A type of machine learning where an agent learns to make decisions by taking actions in an environment to maximize cumulative rewards. This approach is often used in robotics and game playing.

- **Computer Vision**
  An interdisciplinary field that enables machines to interpret and understand visual information from the world, such as images and videos. Applications include facial recognition, object detection, and image classification.

- **Speech Recognition**
  The ability of a computer system to recognize spoken language and convert it into text. This technology is used in virtual assistants, transcription services, and voice-controlled applications.

- **Chatbot**
  An AI application designed to simulate conversation with human users through text or voice interactions. Chatbots are commonly used in customer service to provide information and assistance.

- **Sentiment Analysis**
  The use of NLP techniques to determine the sentiment expressed in a piece of text, such as positive, negative, or neutral feelings. This analysis is often applied in social media monitoring and customer feedback evaluation.

- **Synthetic Data/Data Augmentation**
  Data generated artificially rather than obtained from real-world events. Synthetic data is often used for training AI models when real data is scarce or sensitive.

- **Overfitting**
  A modeling error that occurs when a machine learning model learns the training data too well, including noise and outliers, resulting in poor performance on unseen data.

- **Large Language Model (LLM):**

  A type of AI model trained on vast amounts of text data to understand and generate human-like language, capable of tasks like translation, summarization, and question-answering.

- **Generative AI:**

  A category of AI algorithms that can generate new content, such as text, images, or music, by learning from existing data.

Differences and relationships among AI, ML, and DL.

Definitions

- **Artificial Intelligence (AI)**
  AI is the broadest category that encompasses any technique enabling machines to mimic human intelligence. This includes reasoning, learning, problem-solving, perception, and language understanding. AI systems can be rule-based or data-driven and can operate in various domains such as robotics, natural language processing, and computer vision.

- **Machine Learning (ML)**
  ML is a subset of AI that focuses on algorithms and statistical models that allow computers to learn from data without explicit programming. ML emphasizes the ability of machines to improve their performance based on experience, making it possible for them to identify patterns and make predictions.

- **Deep Learning (DL)**
  DL is a further subset of ML that utilizes neural networks with multiple layers (deep neural networks) to analyze large amounts of unstructured data. DL models automatically learn hierarchical representations of data, allowing them to recognize complex patterns like images or speech without manual feature extraction.

Differences

Aspect	Artificial Intelligence (AI)	Machine Learning (ML)	Deep Learning (DL)
Scope	Broad field encompassing all intelligent systems	Subset of AI focused on learning from data	Subset of ML using deep neural networks
Data Handling	Can work with structured or unstructured data	Primarily works with structured data	Excels at handling large volumes of unstructured data
Learning Method	Can be rule-based or learning-based	Learns from data through algorithms	Learns complex patterns through layered neural networks
Complexity	Varies widely depending on the application	Generally less complex than DL	Highly complex due to multiple processing layers
Applications	Virtual assistants, recommendation systems	Spam detection, predictive analytics	Image recognition, natural language processing

Python Basics

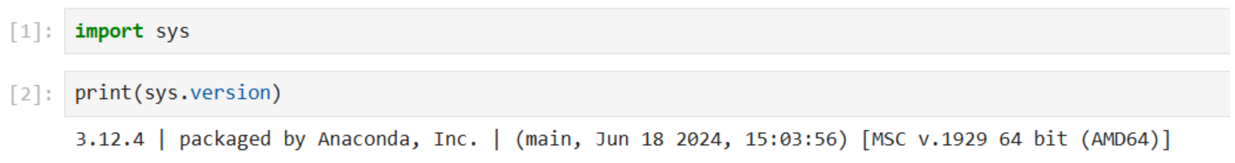
What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Python Version



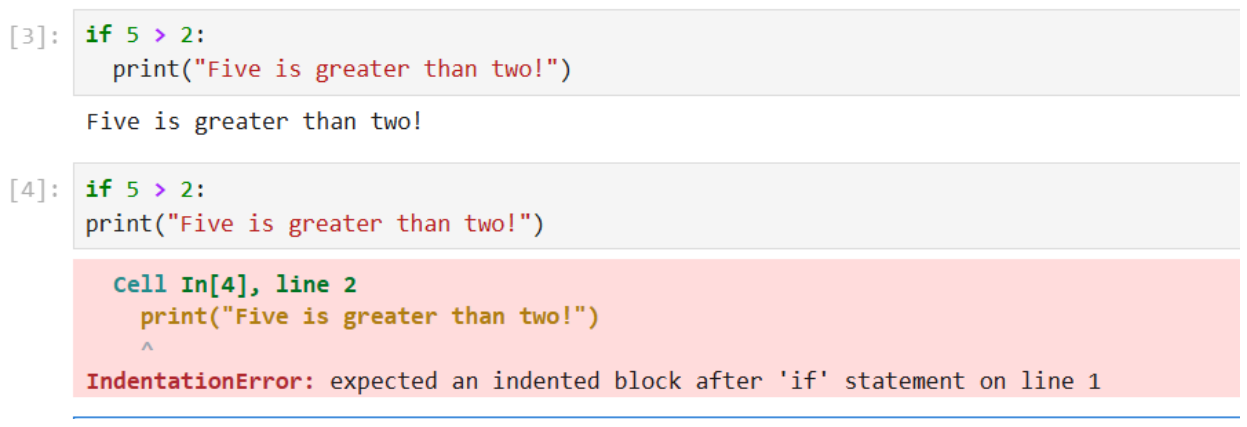
```
[1]: import sys
```

```
[2]: print(sys.version)
```

```
3.12.4 | packaged by Anaconda, Inc. | (main, Jun 18 2024, 15:03:56) [MSC v.1929 64 bit (AMD64)]
```

Python Syntax

Indentation refers to the spaces at the beginning of a code line.Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.Python uses indentation to indicate a block of code.



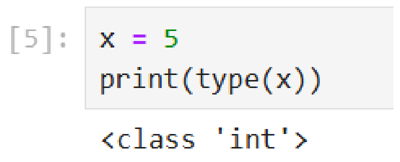
```
[3]: if 5 > 2:
       print("Five is greater than two!")

     Five is greater than two!

[4]: if 5 > 2:
     print("Five is greater than two!")

       Cell In[4], line 2
         print("Five is greater than two!")
         ^
     IndentationError: expected an indented block after 'if' statement on line 1
```

Data Types:

Data Type	Description	Syntax Example	Mutability	Use Cases
List	Ordered collection; allows duplicates.	my_list = [1, 2, 3]	Mutable	Storing items in order (e.g., tasks).
Tuple	Ordered collection; immutable; allows duplicates.	my_tuple = (1, 2, 3)	Immutable	Fixed collections (e.g., coordinates).

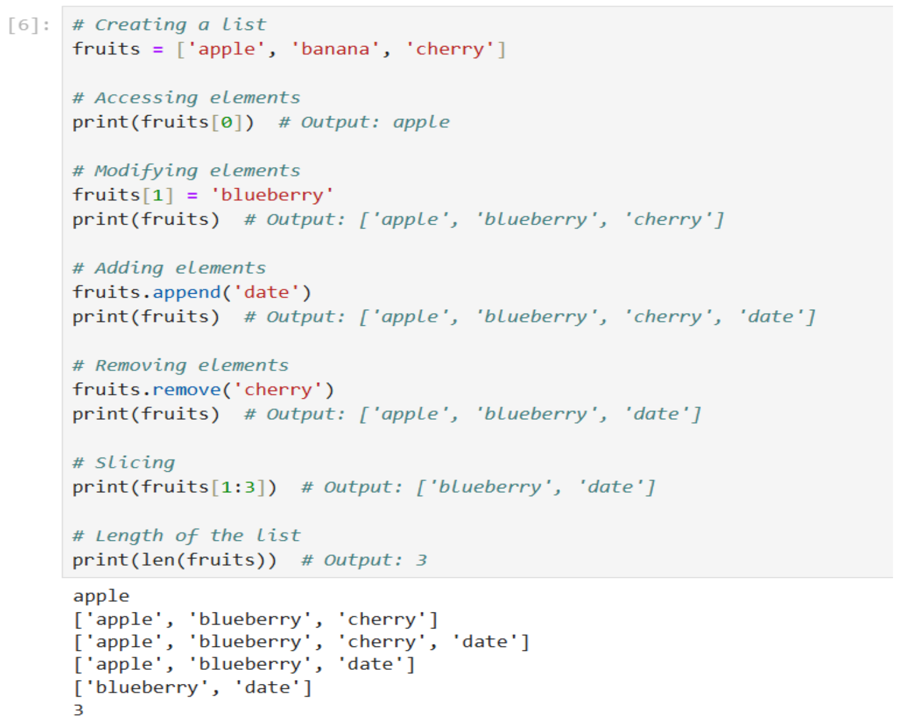
Dictionary	Collection of key-value pairs; keys must be unique.	my_dict = {'key': 'value'}	Mutable	Storing data accessed by keys (e.g., user info).
Set	Unordered collection of unique items.	my_set = {1, 2, 3}	Mutable	Unique items (e.g., tags).

Getting the Data Type



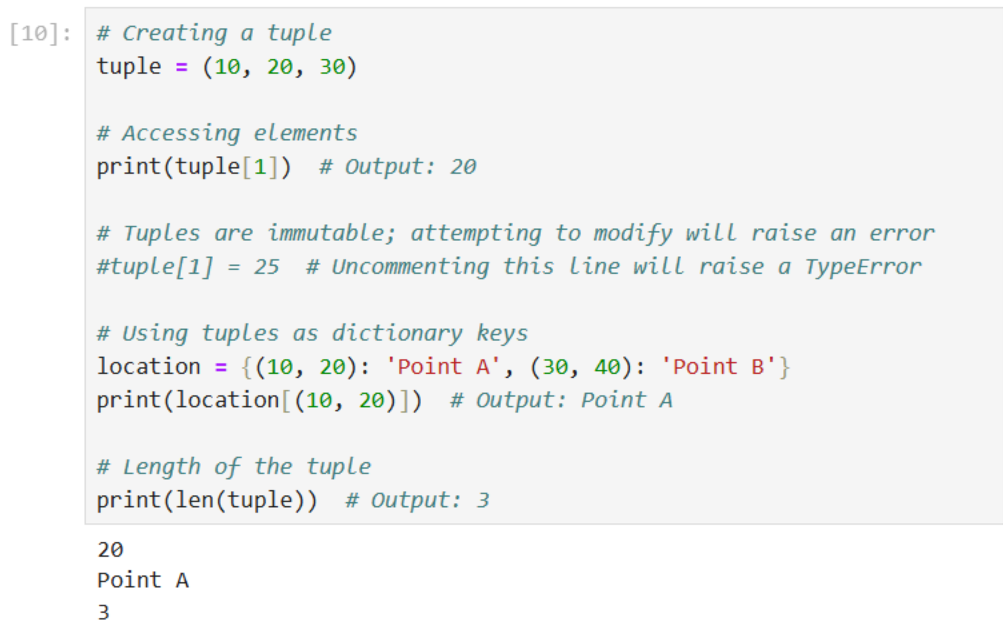
```
[5]: x = 5
     print(type(x))

     <class 'int'>
```

List :



```
[6]: # Creating a list
     fruits = ['apple', 'banana', 'cherry']

     # Accessing elements
     print(fruits[0])  # Output: apple

     # Modifying elements
     fruits[1] = 'blueberry'
     print(fruits)  # Output: ['apple', 'blueberry', 'cherry']

     # Adding elements
     fruits.append('date')
     print(fruits)  # Output: ['apple', 'blueberry', 'cherry', 'date']

     # Removing elements
     fruits.remove('cherry')
     print(fruits)  # Output: ['apple', 'blueberry', 'date']

     # Slicing
     print(fruits[1:3])  # Output: ['blueberry', 'date']

     # Length of the list
     print(len(fruits))  # Output: 3

     apple
     ['apple', 'blueberry', 'cherry']
     ['apple', 'blueberry', 'cherry', 'date']
     ['apple', 'blueberry', 'date']
     ['blueberry', 'date']
     3
```

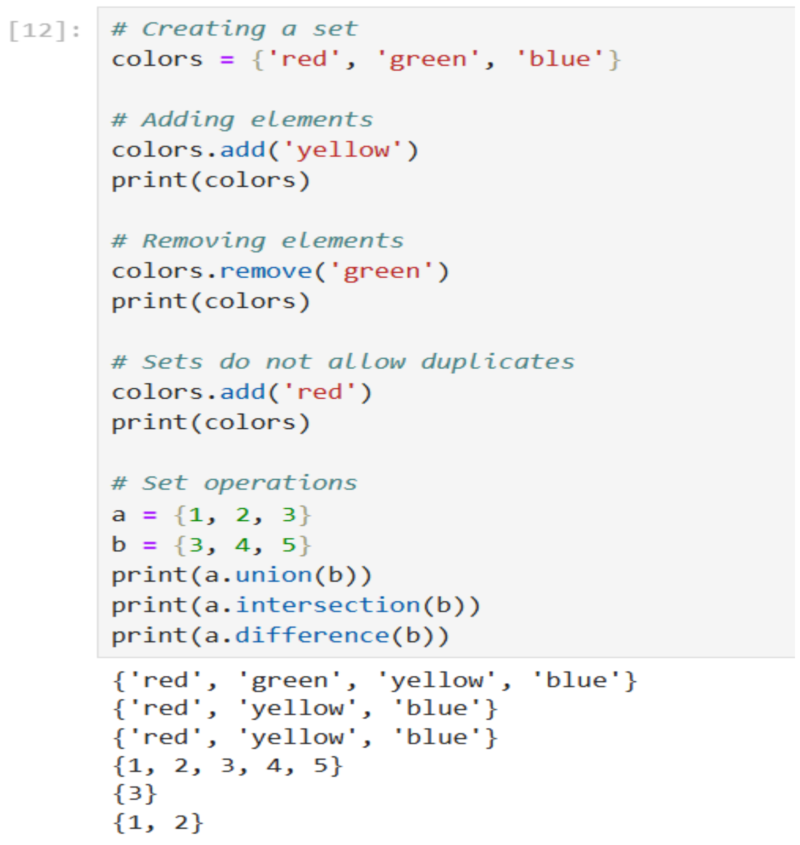
Tuple :



```
[10]: # Creating a tuple
      tuple = (10, 20, 30)

      # Accessing elements
      print(tuple[1])  # Output: 20

      # Tuples are immutable; attempting to modify will raise an error
      #tuple[1] = 25  # Uncommenting this line will raise a TypeError

      # Using tuples as dictionary keys
      location = {(10, 20): 'Point A', (30, 40): 'Point B'}
      print(location[(10, 20)])  # Output: Point A

      # Length of the tuple
      print(len(tuple))  # Output: 3
```

```
20
Point A
3
```

Set :



```
[12]: # Creating a set
      colors = {'red', 'green', 'blue'}

      # Adding elements
      colors.add('yellow')
      print(colors)

      # Removing elements
      colors.remove('green')
      print(colors)

      # Sets do not allow duplicates
      colors.add('red')
      print(colors)

      # Set operations
      a = {1, 2, 3}
      b = {3, 4, 5}
      print(a.union(b))
      print(a.intersection(b))
      print(a.difference(b))
```

```
{'red', 'green', 'yellow', 'blue'}
{'red', 'yellow', 'blue'}
{'red', 'yellow', 'blue'}
{1, 2, 3, 4, 5}
{3}
{1, 2}
```

Dictionary :

```
[13]: # Creating a dictionary
person = {'name': 'Alice', 'age': 30, 'city': 'New York'}

# Accessing values
print(person['name'])

# Adding key-value pairs
person['email'] = 'alice@example.com'
print(person)

# Removing key-value pairs
del person['age']
print(person)

# Iterating through a dictionary
for key, value in person.items():
    print(f'{key}: {value}')

# Length of the dictionary
print(len(person))  # Output: 3
```

```
Alice
{'name': 'Alice', 'age': 30, 'city': 'New York', 'email': 'alice@example.com'}
{'name': 'Alice', 'city': 'New York', 'email': 'alice@example.com'}
name: Alice
city: New York
email: alice@example.com
3
```

Control Flow: Loops and Conditionals

If ... Else :

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.

If

An "if statement" is written by using the if keyword.

```
[1]: a = 33
     b = 200
     if b > a:
       print("b is greater than a")
```

```
b is greater than a
```

Elif

The elif keyword is Python's way of saying "if the previous conditions were not true, then try this condition".

```
[2]: a = 33
     b = 33
     if b > a:
       print("b is greater than a")
     elif a == b:
       print("a and b are equal")
```

```
a and b are equal
```

Else

The else keyword catches anything which isn't caught by the preceding conditions.

```
[3]: a = 200
     b = 33
     if b > a:
       print("b is greater than a")
     elif a == b:
       print("a and b are equal")
     else:
       print("a is greater than b")
```

```
a is greater than b
```

Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line

```
[4]: a = 2
     b = 330
     print("A") if a > b else print("B")
```

```
B
```

And

The and keyword is a logical operator, and is used to combine conditional statements

```
[5]: a = 200
     b = 33
     c = 500
     if a > b and c > a:
       print("Both conditions are True")
```

```
Both conditions are True
```

Nested If

You can have if statements inside if statements, this is called *nested* if statements.

```
[6]: x = 41

     if x > 10:
       print("Above ten,")
       if x > 20:
         print("and also above 20!")
       else:
         print("but not above 20.")
```

```
Above ten,
and also above 20!
```

While Loops

With the while loop we can execute a set of statements as long as a condition is true.

```
[7]: i = 1
     while i < 6:
       print(i)
       i += 1
```

```
1
2
3
4
5
```

The break Statement

With the break statement we can stop the loop even if the while condition is true.

For Loop

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

```
[8]: fruits = ["apple", "banana", "cherry"]
     for x in fruits:
       print(x)
```

```
apple
banana
cherry
```

The range() Function

To loop through a set of code a specified number of times, we can use the range() function,

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6).

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3).

```
[11]: for x in range(4):
        print(x)
      print("-" * 50)

      for x in range(2, 4):
        print(x)
      print("-" * 50)

      for x in range(2, 20, 4):
        print(x)
```

```
0
1
2
3
------------------------------
2
3
------------------------------
2
6
10
14
18
```

Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop".

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

Python Functions :

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

In Python a function is defined using the def keyword.

To call a function, use the function name followed by parenthesis.

```
def my_function():
  print("Hello from a function")

my_function()
```

```
Hello from a function
```

*args

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

This way the function will receive a *tuple* of arguments, and can access the items accordingly.

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

```
The youngest child is Linus
```

****kwargs**

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.

This way the function will receive a *dictionary* of arguments, and can access the items accordingly.

```python
def print_details(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

print_details(name="Alice", age=25, city="New York")
```

```
name: Alice
age: 25
city: New York
```

Recursion

Python also accepts function recursion, which means a defined function can call itself.

```python
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

number = 5
result = factorial(number)
print(f"The factorial of {number} is {result}")
```

```
The factorial of 5 is 120
```

Lambda

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

The power of lambda is better shown when you use them as an anonymous function inside another function.

Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number.

```python
def multiplier(n):
    return lambda x: x * n

triple = multiplier(3)
print(triple(5))
```

```
15
```

Classes and Objects:

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

To create a class, use the keyword class.

```
class MyClass:

  x = 5
```

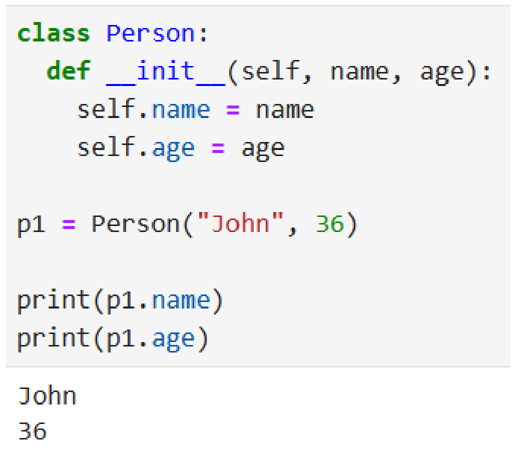
Now we can use the class named MyClass to create objects.

```
p1 = MyClass()

print(p1.x)
```

__init__() Function

All classes have a function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created.



```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

```
John
36
```

__str__() Function

The __str__() function controls what should be returned when the class object is represented as a string.

If the __str__() function is not set, the string representation of the object is returned.

```python
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def __str__(self):
    return f"{self.name}({self.age})"

p1 = Person("John", 36)

print(p1)
```

```
John(36)
```

Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object.

Let us create a method in the Person class.

```python
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

```
Hello my name is John
```

The self Parameter

The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

It does not have to be named self, you can call it whatever you like, but it has to be the first parameter of any function in the class.

```python
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

```
Hello my name is John
```

Python Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

```python
class Person:

  def __init__(self, fname, lname):

    self.firstname = fname

     self.lastname = lname

  def printname(self):

     print(self.firstname, self.lastname)
```

```
x = Person("John", "Doe")

x.printname()



class Student(Person):

  Pass

x = Student("Mike", "Olsen")
x.printname()
```

```
John Doe
Mike Olsen
```

So far we have created a child class that inherits the properties and methods from its parent.

We want to add the __init__() function to the child class (instead of the pass keyword).

```
class Student(Person):

  def __init__(self, fname, lname):

    Person.__init__(self, fname, lname)
```

super() Function

Python also has a super() function that will make the child class inherit all the methods and properties from its parent.

```
class Student(Person):

  def __init__(self, fname, lname):

    super().__init__(fname, lname)
```

Polymorphism

Polymorphism is often used in Class methods, where we can have multiple classes with the same method name.

For example, say we have three classes: Car, Boat, and Plane, and they all have a method called move().

```python
class Car:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Drive!")

class Boat:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Sail!")

class Plane:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Fly!")

car1 = Car("Ford", "Mustang")       #Create a Car object
boat1 = Boat("Ibiza", "Touring 20") #Create a Boat object
plane1 = Plane("Boeing", "747")     #Create a Plane object

for x in (car1, boat1, plane1):
  x.move()
```

```
Drive!
Sail!
Fly!
```

Inheritance Class Polymorphism

What about classes with child classes with the same name? Can we use polymorphism there?

Yes. If we use the example above and make a parent class called Vehicle, and make Car, Boat, Plane child classes of Vehicle, the child classes inherits the Vehicle methods, but can override them.

```
class Vehicle:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Move!")

class Car(Vehicle):
  pass

class Boat(Vehicle):
  def move(self):
    print("Sail!")

class Plane(Vehicle):
  def move(self):
    print("Fly!")

car1 = Car("Ford", "Mustang")       #Create a Car object
boat1 = Boat("Ibiza", "Touring 20") #Create a Boat object
plane1 = Plane("Boeing", "747")     #Create a Plane object

for x in (car1, boat1, plane1):
  print(x.brand)
  print(x.model)
  x.move()
```

```
Ford
Mustang
Move!
Ibiza
Touring 20
Sail!
Boeing
747
Fly!
```

Error Handling :

Error handling in Python is managed using try, except, else, and finally blocks. These structures allow you to handle runtime errors gracefully without crashing your program.

The try block lets you test a block of code for errors.

The except block lets you handle the error.

The else block lets you execute code when there is no error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

```
Something went wrong
The 'try except' is finished
```

```
try:
  f = open("demofile.txt")
  try:
    f.write("Lorum Ipsum")
  except:
    print("Something went wrong when writing to the file")
  finally:
    f.close()
except:
  print("Something went wrong when opening the file")
```

```
Something went wrong when opening the file
```

Raise an exception

As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the raise keyword.

You can define what kind of error to raise, and the text to print to the user.

```python
x = -1

if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
Cell In[3], line 4
      1 x = -1
      3 if x < 0:
----> 4   raise Exception("Sorry, no numbers below zero")

Exception: Sorry, no numbers below zero
```

```python
x = "hello"

if not type(x) is int:
  raise TypeError("Only integers are allowed")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[4], line 4
      1 x = "hello"
      3 if not type(x) is int:
----> 4   raise TypeError("Only integers are allowed")

TypeError: Only integers are allowed
```