Explain the key features of Python that make it a popular choice for programming.

- It is easy to use and open source platform.
- Python is cross-platform, meaning that Python code can run on various operating systems like Windows, macOS, Linux, and more, without modification.
- Python is dynamically typed, meaning that you don't need to declare variable types explicitly. The type is determined at runtime, which can make coding faster and more flexible.
- Python uses indentation to define code blocks, which promotes clean and consistent code.
- Python fully supports object-oriented concepts like classes, inheritance, polymorphism, and encapsulation.

Describe the role of predefined keywords in Python and provide example of how they are used in a programming.

Predefined keywords in Python are reserved words that have special meaning in the language. They serve as the building blocks of Python's syntax, defining the structure, control flow, and logic of the program. Keywords cannot be used as identifiers (like variable names, function names, etc.) because they are reserved for specific language constructs.

In [1]:
```
# for, if, else, in, def, class, elif, break, pass, continue, while etc are the exampl

# suppose we want to run for loop then to create the code we use for keyword

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

```
0
1
2
3
4
```

Compare and contrast mutable and immutable objects in Python with examples

Mutable objects are those objects in which we can make changes, we can modity the elements. for ex, list, dictionary, set etc are the mutable objects.

In [2]:
```
li = [1,2,3,4]
li[1]=10
print(li)
```

```
[1, 10, 3, 4]
```

In [4]:
```
li = [1,2,3,4]
li.append(120)
print(li)
```

```
[1, 2, 3, 4, 120]
```

In [5]:
```
dic = {'a':1, 'b':2, 'c':3}
dic['a']=10
print(dic)
```

```
{'a': 10, 'b': 2, 'c': 3}
```

In [6]:
```
# Immutable objects are those in which we cant make any changes once it is created, fo
tu=(1,2,3,5)
tu.append(10)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[6], line 3
      1 # Immutable objects are those in which we cant make any changes once it is cr
eated, for ex, tuple, string
      2 tu=(1,2,3,5)
----> 3 tu.append(10)

AttributeError: 'tuple' object has no attribute 'append'
```

Discuss the different types of operators in Python and provide examples of how they are used

In Python, **operators are special symbols that perform operations on variables and values.** Python supports various types of operators, each serving different purposes.

1. Arithmetic Operators

Arithmetic operators are used to perform basic mathematical operations.

a= 2, b= 4

- Addition (+): Adds two numbers. res = a+b
- Subtraction (-): Subtracts one number from another. res = a-b
- Multiplication (*): Multiplies two numbers res =ab
- Division (/): Divides one number by another (result is a float).
- Floor Division (//): Divides one number by another and rounds down to the nearest integer
- Modulus (%): Returns the remainder of the division
- Exponentiation (**): Raises one number to the power of another

2. Comparison Operators

Comparison operators are used to compare two values and return a Boolean value (True or False).

- Equal (==): Checks if two values are equal. res= (a==b)
- Not Equal (!=): Checks if two values are not equal. res = (a!=b)

- Greater Than (>): Checks if the left value is greater than the right value. res = (a>b)
- Less Than (<): Checks if the left value is less than the right value
- Greater Than or Equal To (>=): Checks if the left value is greater than or equal to the right value.
- Less Than or Equal To (<=): Checks if the left value is less than or equal to the right value.

3. Assignment Operators

Assignment operators are used to assign values to variables.

- Assignment (=): Assigns the value on the right to the variable on the left.
- Add and Assign (+=): Adds the right value to the left variable and assigns the result to the left variable.
- Subtract and Assign (-=): Subtracts the right value from the left variable and assigns the result to the left variable.
- Multiply and Assign (*=): Multiplies the left variable by the right value and assigns the result to the left variable.

we can use division and assign, modulus and assign like the same as above.

4. Logical Operators

Logical operators are used to perform logical operations on Boolean values.

- AND (and): Returns True if both operands are true.

  ```
  result = (a > b) and (b > 0)  # result is True
  ```
- OR (or): Returns True if at least one operand is true.

  ```
  result = (a < b) or (b > 0)  # result is True
  ```
- NOT (not): Returns True if the operand is false.

  ```
  result = not(a > b)  # result is False
  ```

5. Membership Operators

Membership operators are used to test if a value is a member of a sequence (like a list, tuple, or string).

- IN (in): Returns True if the value is found in the sequence.

  ```
  fruits = ["apple", "banana", "cherry"]
  result = "banana" in fruits  # result is True
  ```
- NOT IN (not in): Returns True if the value is not found in the sequence.

  ```
  result = "grape" not in fruits  # result is True
  ```

Explain the concept of type casting in python with example

Type casting in Python refers to the process of converting one data type into another. This is useful when you need to perform operations on variables that require them to be of a specific type. Python supports both implicit and explicit type casting.

1. Implicit Type Casting

- Implicit type casting is performed automatically by Python when it converts a smaller data type to a larger data type to avoid data loss.
- Python automatically converts an integer to a float when a float is involved in an arithmetic operation.

In [12]:
```
a = 3
b=5.2
res = a+b
print(res)
```

```
8.2
```

2. Explicit type casting

Explicit type casting is done manually by the programmer using Python's built-in functions to convert one data type to another.

The most common functions used for type casting are:

- int(): Converts a value to an integer.
- float(): Converts a value to a float.
- str(): Converts a value to a string.
- list(): Converts a sequence (like a tuple or a string) to a list.
- tuple(): Converts a sequence (like a list or a string) to a tuple.

In [20]:
```
str1 = '1000'
print(type(str1))
val = 5.6
res = int(str1)+val
print(res, type(res))
```

```
<class 'str'>
1005.6 <class 'float'>
```

How do conditional statement works in python? illustrate with example.

Conditional statements in Python are used to execute a block of code based on whether a condition is True or False. The primary conditional statements in Python are if, elif, and else.

In [26]:
```
for i in range(30):
    if i==5:
        break
    else:
        print(i)
```

```
        continue

# it will break the loop if it encounters with 5
```

```
0
1
2
3
4
```

In [27]:
```
even=[]
odd=[]
for i in range(20):
    if i%2==0:
        even.append(i)
    else:
        odd.append(i)

print(even)
print(odd)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

Describe the different types of loops in python and their use cases with examples.

In Python, loops are used to repeatedly execute a block of code as long as a condition is met. There are two main types of loops in Python: for loops and while loops. Each has its specific use cases depending on the scenario.

for loop:

The for loop is used to iterate over a sequence (like a list, tuple, dictionary, set, or string) or any other iterable object. It executes the block of code for each item in the sequence.

In [30]:
```
# for loop
li = ['banana','apple','mango']
for i in li:
    print(i)
```

```
banana
apple
mango
```

In [40]:
```
dic = {'banana':2, 'apple':4, 'mango':6}
for i in dic.items():
    print(i)
```

```
('banana', 2)
('apple', 4)
('mango', 6)
```

while loop:

The while loop repeatedly executes a block of code as long as a given condition is True. The loop will continue running until the condition becomes False.

In [1]:
```
# while loop
count=1
while count<=5:
    print(count)
    count+=1
```

```
1
2
3
4
5
```

In [ ]:
```

```

In [ ]:
```

```