Variables: Variables are containers for storing values. In Python, you can declare a variable by assigning a value to it.

# Example:

x = 10

y = 20

z = x + y

print(z) # Output: 30

Data Types: Python has several built-in data types, including numbers (integer, float), strings, lists, tuples, and dictionaries.

# Example:

# Numbers

a = 10

b = 20.5

# Strings

name = "John Doe"

# Lists

fruits = ["apple", "banana", "cherry"]

# Tuples

coordinates = (10, 20)

# Dictionaries

person = {

"name": "John Doe",

"age": 30,

"city": "New York"

}

Operators: Operators are used to perform operations on variables and values. Python has various types of operators such as arithmetic, comparison, assignment, etc.

# Example:

# Arithmetic operators

x = 10

y = 20

z = x + y # Output: 30

# Comparison operators

a = 10

b = 20

print(a == b) # Output: False

print(a != b) # Output: True

# Assignment operators

x = 10

x += 5 # x is now 15

Control Flow: Control flow statements determine the order in which the code is executed. Python has if-else, for and while loops.

# Example:

# if-else statement

x = 10

if x > 0:

print("x is positive")

else:

print("x is negative") # Output: x is positive

# for loop

fruits = ["apple", "banana", "cherry"]

for fruit in fruits:

print(fruit) # Output: apple, banana, cherry

# while loop

x = 0

while x < 5:

print(x)

x += 1 # Output: 0, 1, 2, 3, 4

Functions: Functions are reusable blocks of code that perform a specific task. You can define a function using the def keyword.

# Example:

def greet(name):

return "Hello, " + name

print(greet("John")) # Output: Hello, John

Classes and Objects: Classes define objects and their behavior in an Object-Oriented Programming (OOP) paradigm.

# Example:

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def greet(self):

return "Hello, my name is " + self.name

p = Person("John Doe", 30)

print(p.greet()) # Output: Hello, my name is John Doe

Lists in Python are ordered collections of items, which can be of different data types, including numbers, strings, and even other lists. You can create a list in Python by enclosing a comma-separated sequence of items in square brackets ([]).

Here's an example:

# Creating a list of integers

numbers = [1, 2, 3, 4, 5]

# Creating a list of strings

fruits = ["apple", "banana", "cherry"]

# Creating a mixed list

mixed = [1, "apple", 3.14, [1, 2, 3]]

You can access items in a list by using their index, which starts from 0. For example:

# Accessing the first item in the list

print(fruits[0]) # Output: apple

# Accessing the last item in the list

print(fruits[-1]) # Output: cherry

In addition to accessing items, you can also modify a list by adding or removing items. Here's how:

# Adding items to a list

fruits.append("orange")

# Removing items from a list

fruits.remove("banana")

Lists also support various built-in methods, such as sort() for sorting the items in the list, len() for finding the length of the list, and reverse() for reversing the order of the items in the list.

Here's an example:

# Sorting a list

numbers.sort()

print(numbers) # Output: [1, 2, 3, 4, 5]

# Finding the length of a list

print(len(fruits)) # Output: 2

# Reversing a list

fruits.reverse()

print(fruits) # Output: ['cherry', 'apple']

Sets in Python are unordered collections of unique items. Unlike lists and tuples, sets do not allow duplicates, and they use curly braces ({}) to define the set. You can also use the set() built-in function to create a set.

Here's an example:

# Creating a set using curly braces

fruits = {"apple", "banana", "cherry"}

# Creating a set using the set() function

numbers = set([1, 2, 3, 4, 5, 5, 5])

In a set, each item is unique, so duplicates are automatically removed. In the example above, the set numbers only contains the numbers 1, 2, 3, 4, and 5, even though 5 was specified three times.

You can perform various operations on sets, such as union, intersection, and difference. Here's how:

# Union of two sets

a = {1, 2, 3}

b = {3, 4, 5}

c = a | b

print(c) # Output: {1, 2, 3, 4, 5}

# Intersection of two sets

a = {1, 2, 3}

b = {2, 3, 4}

c = a & b

print(c) # Output: {2, 3}

# Difference of two sets

a = {1, 2, 3}

b = {2, 3, 4}

c = a - b

print(c) # Output: {1}

Sets are useful when you need to store unique elements and perform set operations such as union, intersection, and difference. They do not support indexing or slicing, so you cannot access elements in a set by index. However, you can check if an item is in a set using the in keyword.

Here's an example:

# Checking if an item is in a set

fruits = {"apple", "banana", "cherry"}

print("apple" in fruits) # Output: True

print("orange" in fruits) # Output: False