# Python-learning-Day1

1. Comments in Python

Single-line comments: Use the # symbol. Everything after # on that line is ignored by the interpreter.

# This is a single-line comment

print("Hello, World!") # This is an inline comment

Multi-line comments: Technically, Python doesn’t have multi-line comment syntax, but you can use triple quotes (''' or """) as a workaround.

"""

This is a multi-line comment

that spans several lines.

"""

print("Multi-line comments above!")

3. Keywords in Python

Definition: Reserved words that have special meaning in Python. They are part of the syntax and cannot be used as identifiers (variable names).

Examples: if, else, elif, while, for, def, return, True, False, import, class, try, except, finally, with, as, break, continue, pass, global, lambda, nonlocal, yield, etc.

if True:

print("This is a keyword example.")

Note: You can view all keywords using the keyword module:

import keyword

print(keyword.kwlist)

3.1 Identifiers in Python

Definition: Names used to identify variables, functions, classes, modules, etc.

Rules:

It can contain letters (A–Z, a–z), digits (0–9), and underscores (\_).

It cannot start with a digit.

It cannot be a Python keyword.

Case-sensitive (Variable and variable are different).

name = "John" # valid identifier

\_age = 25 # valid identifier

1st\_number = 100 # invalid identifier (cannot start with a digit)

Here's an overview of the requested Python concepts:

1. Comments in Python

Single-line comments: Use the # symbol. Everything after # on that line is ignored by the interpreter.

python

Copy

Edit

# This is a single-line comment

print("Hello, World!") # This is an inline comment

Multi-line comments: Technically, Python doesn’t have multi-line comment syntax, but you can use triple quotes (''' or """) as a workaround.

python

Copy

Edit

"""

This is a multi-line comment

that spans several lines.

"""

print("Multi-line comments above!")

2. Keywords in Python

Definition: Reserved words that have special meaning in Python. They are part of the syntax and cannot be used as identifiers (variable names).

Examples: if, else, elif, while, for, def, return, True, False, import, class, try, except, finally, with, as, break, continue, pass, global, lambda, nonlocal, yield, etc.

python

Copy

Edit

if True:

print("This is a keyword example.")

Note: You can view all keywords using the keyword module:

python

Copy

Edit

import keyword

print(keyword.kwlist)

3. Identifiers in Python

Definition: Names used to identify variables, functions, classes, modules, etc.

Rules:

Can contain letters (A–Z, a–z), digits (0–9), and underscores (\_).

Cannot start with a digit.

Cannot be a Python keyword.

Case-sensitive (Variable and variable are different).

python

Copy

Edit

name = "John" # valid identifier

\_age = 25 # valid identifier

1st\_number = 100 # invalid identifier (cannot start with a digit)

4. Operators in Python

a. Arithmetic Operators

Used for mathematical operations:

+ (Addition)

- (Subtraction)

\* (Multiplication)

/ (Division)

// (Floor Division)

% (Modulus - remainder)

\*\* (Exponentiation)

a = 10

b = 3

print(a + b) # 13

print(a - b) # 7

print(a \* b) # 30

print(a / b) # 3.333...

print(a // b) # 3

print(a % b) # 1

print(a \*\* b) # 1000

b. Comparison Operators

Used to compare values, returning True or False:

== (Equal to)

!= (Not equal to)

> (Greater than)

< (Less than)

>= (Greater than or equal to)

<= (Less than or equal to)

x = 5

y = 10

print(x == y) # False

print(x != y) # True

print(x > y) # False

print(x < y) # True

print(x >= 5) # True

print(y <= 10) # True

Here's an overview of the requested Python concepts:

1. Comments in Python

Single-line comments: Use the # symbol. Everything after # on that line is ignored by the interpreter.

python

Copy

Edit

# This is a single-line comment

print("Hello, World!") # This is an inline comment

Multi-line comments: Technically, Python doesn’t have multi-line comment syntax, but you can use triple quotes (''' or """) as a workaround.

python

Copy

Edit

"""

This is a multi-line comment

that spans several lines.

"""

print("Multi-line comments above!")

2. Keywords in Python

Definition: Reserved words that have special meaning in Python. They are part of the syntax and cannot be used as identifiers (variable names).

Examples: if, else, elif, while, for, def, return, True, False, import, class, try, except, finally, with, as, break, continue, pass, global, lambda, nonlocal, yield, etc.

python

Copy

Edit

if True:

print("This is a keyword example.")

Note: You can view all keywords using the keyword module:

python

Copy

Edit

import keyword

print(keyword.kwlist)

3. Identifiers in Python

Definition: Names used to identify variables, functions, classes, modules, etc.

Rules:

Can contain letters (A–Z, a–z), digits (0–9), and underscores (\_).

Cannot start with a digit.

Cannot be a Python keyword.

Case-sensitive (Variable and variable are different).

python

Copy

Edit

name = "John" # valid identifier

\_age = 25 # valid identifier

1st\_number = 100 # invalid identifier (cannot start with a digit)

4. Operators in Python

a. Arithmetic Operators

Used for mathematical operations:

+ (Addition)

- (Subtraction)

\* (Multiplication)

/ (Division)

// (Floor Division)

% (Modulus - remainder)

\*\* (Exponentiation)

python

Copy

Edit

a = 10

b = 3

print(a + b) # 13

print(a - b) # 7

print(a \* b) # 30

print(a / b) # 3.333...

print(a // b) # 3

print(a % b) # 1

print(a \*\* b) # 1000

b. Comparison Operators

Used to compare values, returning True or False:

== (Equal to)

!= (Not equal to)

> (Greater than)

< (Less than)

>= (Greater than or equal to)

<= (Less than or equal to)

python

Copy

Edit

x = 5

y = 10

print(x == y) # False

print(x != y) # True

print(x > y) # False

print(x < y) # True

print(x >= 5) # True

print(y <= 10) # True

c. Assignment Operators

Used to assign values to variables:

= (Simple assignment)

+= (Add and assign)

-= (Subtract and assign)

\*= (Multiply and assign)

/= (Divide and assign)

//= (Floor divide and assign)

%= (Modulus and assign)

\*\*= (Exponentiate and assign)

a = 5 # Assign 5 to a

a += 3 # Equivalent to a = a + 3 (a becomes 8)

a \*= 2 # Equivalent to a = a \* 2 (a becomes 16)

a -= 4 # Equivalent to a = a - 4 (a becomes 12)

a /= 3 # Equivalent to a = a / 3 (a becomes 4.0)

# Python-learning-Day2

1. Logical Operators

Logical operators are used to combine conditional statements:

and= Returns True if both conditions are true (5 > 3) and (4 < 6) -> True

or= Returns True if at least one condition is true (5 < 3) or (4 < 6) -> True

not= Reverses the result (negation) not(5 > 3) -> False

code(syntax):-

x = 10

y = 5

# Logical AND

print(x > 5 and y < 10) # True

# Logical OR

print(x < 5 or y < 10) # True

# Logical NOT

print(not(x == 10)) # False

2. Membership Operators

Membership operators test for membership in sequences like lists, tuples, strings, etc.

in= Returns True if a value exists in the sequence 'a' in 'apple' -> True

not in= Returns True if a value does NOT exist in the sequence 3 not in [1, 2, 4] -> True

code(syntax):-

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

# Membership Test

print("banana" in fruits) # True

print("grape" not in fruits) # True

3. Identity Operators

Identity operators compare the memory location of two objects (whether they are the same object in memory).

is= Returns True if both variables point to the same object a is b

is not= Returns True if both variables point to different objects a is not b

code(syntax):-

a = [1, 2, 3]

b = a

c = [1, 2, 3]

print(a is b) # True (same object)

print(a is c) # False (different objects with same content)

print(a is not c) # True

Here's a breakdown of logical operators, membership operators, identity operators, and conditional statements (if, else, elif) in Python:

1. Logical Operators

Logical operators are used to combine conditional statements:

Operator Description Example

and Returns True if both conditions are true (5 > 3) and (4 < 6) → True

or Returns True if at least one condition is true (5 < 3) or (4 < 6) → True

not Reverses the result (negation) not(5 > 3) → False

Example:

python

Copy

Edit

x = 10

y = 5

# Logical AND

print(x > 5 and y < 10) # True

# Logical OR

print(x < 5 or y < 10) # True

# Logical NOT

print(not(x == 10)) # False

2. Membership Operators

Membership operators test for membership in sequences like lists, tuples, strings, etc.

Operator Description Example

in Returns True if a value exists in the sequence 'a' in 'apple' → True

not in Returns True if a value does NOT exist in the sequence 3 not in [1, 2, 4] → True

Example:

python

Copy

Edit

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

# Membership Test

print("banana" in fruits) # True

print("grape" not in fruits) # True

3. Identity Operators

Identity operators compare the memory location of two objects (whether they are the same object in memory).

Operator Description Example

is Returns True if both variables point to the same object a is b

is not Returns True if both variables point to different objects a is not b

Example:

python

Copy

Edit

a = [1, 2, 3]

b = a

c = [1, 2, 3]

print(a is b) # True (same object)

print(a is c) # False (different objects with same content)

print(a is not c) # True

4. Conditional Statements (if, elif, else)

Conditional statements allow you to execute code based on specific conditions.

Syntax:-

if condition:

# Code block (executes if condition is True)

elif another\_condition:

# Executes if 'if' was False, but this is True

else:

# Executes if all previous conditions were False

code:-

age = 18

if age < 18:

print("You are a minor.")

elif age == 18:

print("You just became an adult!")

else:

print("You are an adult.")

# Python-learning-Day3

1. Nested if-else Statements

Nested if-else statements are if statements inside another if or else block. They help handle multiple conditions.

Code:-

x = 15

if x > 10:

if x < 20:

print("x is between 10 and 20")

else:

print("x is greater than or equal to 20")

else:

print("x is 10 or less")

2. String Formatting

String formatting allows you to insert variables into strings.

Using f-strings (Python 3.6+):

name = "Alice"

age = 25

print(f"My name is {name} and I am {age} years old.")

Using format() method:

print("My name is {} and I am {} years old.".format(name, age))

Using % operator (older style):

print("My name is %s and I am %d years old." % (name, age))

1. String Indexing

In Python, strings are sequences of characters, and each character has a position (index).

Positive Indexing: Starts from 0 (left to right).

Negative Indexing: Starts from -1 (right to left).

Code:-

city="Rajasthan"

city[0]

# Python-learning-Day4

Call by Value vs Call by Reference (Call by Object Reference in Python):-

In Python, arguments are passed by object reference, meaning:

Immutable objects (like int, str, tuple) behave like call by value (modifications inside the function don’t affect the original object).

Mutable objects (like list, dict, set) behave like call by reference (modifications inside the function affect the original object).

Mutable vs Immutable:-

| Type | Mutable (Can Change) | Immutable (Cannot Change) |
| --- | --- | --- |
| Examples | list, dict, set, bytearray | int, float, str, tuple, frozenset, bytes |
| Behavior | Can be modified after creation | Cannot be modified after creation |

For Loop in Python:-

A for loop is used to iterate over sequences (lists, tuples, strings, etc.) or a range of numbers.

Syntax:-

for i in range(1, 6):

print(i)

# Python-learning-Day5

break and continue in Python

In Python, break and continue are control flow statements used within loops to alter their execution.

1. break

Used to exit a loop prematurely.

When break is encountered, the loop terminates immediately, and control moves to the first statement after the loop.

Example of break:

for i in range(1, 6):

if i == 3:

print("Breaking the loop at", i)

break # Exits the loop when i is 3

print(i)

# Output:

# 1

# 2

# Breaking the loop at 3

2. continue

Used to skip the current iteration and move to the next iteration of the loop.

When continue is encountered, the loop does not terminate but skips the remaining code in the current iteration.

Example of continue:

for i in range(1, 6):

if i == 3:

print("Skipping", i)

continue # Skips the rest of the loop body when i is 3

print(i)

# Output:

# 1

# 2

# Skipping 3

# 4

# 5

break vs. continue

| Statement | Effect |
| --- | --- |
| break | Exits the loop completely |
| continue | Skips the current iteration and proceeds with the next iteration |

while Loop in Python

A while loop is used to execute a block of code repeatedly as long as a given condition is True.

Syntax:

while condition:

# Code to execute

The condition is checked before each iteration.

If the condition becomes False, the loop stops.

Example: Simple while Loop

count = 1

while count <= 5:

print("Count:", count)

count += 1 # Increment count to avoid an infinite loop

# Output:

# Count: 1

# Count: 2

# Count: 3

# Count: 4

# Count: 5

| Feature | Description |
| --- | --- |
| while condition: | Executes the loop as long as the condition is True |

|  |  |
| --- | --- |

# Python-learning-Day6

Palindrome Number

A palindrome number is a number that remains the same when its digits are reversed.

s="saras"

num=0

last=len(s)-1

while(num< last ):

if(s[num]!=s[last] ):

print("not palindrome")

break

num+=1

last-=1

else:

print("palindrome")

Armstrong Number

An Armstrong number (or narcissistic number) of order n is a number that is equal to the sum of its own digits each raised to the power n.

n=153

x=n

y=0

total=0

while(n>0):

y=n%10

total=total+y\*\*3

n=n//10

print("n=>",n,"y=>",y,"total=>",total)

if(n==total):

print("armstrong")

else:

print("not armstrong")

# Python-learning-Day7

Nested Loops:

A nested loop in Python is a loop inside another loop. The inner loop executes completely for each iteration of the outer loop. Nested loops are commonly used for working with multi-dimensional data structures like matrices, tables, and grids.

(Day 8, 9(Pattern Assignment),10(Test))

# Python-learning-Day11

List:

A list is a built-in data structure in Python that is used to store multiple items in a single variable. Lists are ordered, mutable (modifiable), and can hold elements of different data types.

1. .append(value)

Adds a single value to the end of the list.

Modifies the list in place.

Only one item can be appended at a time.

2. .extend(iterable)

Adds multiple values from an iterable (list, tuple, set, etc.) to the list.

Modifies the list in place.

The elements of the iterable are unpacked and added individually.

3. .insert(index, value)

Inserts a value at a specific index.

Shifts elements to the right to accommodate the new value.

Slower than .append() and .extend() for large lists.

4. .pop(index=-1)

Removes and returns an element at a given index (default: last element).

Modifies the list in place.

If the index is not provided, the last element is removed.

| Method | Description | Example | Result |
| --- | --- | --- | --- |
| .append(x) | Adds x to the end | list.append(4) | [1, 2, 3, 4] |
| .extend(iterable) | Adds elements from iterable to the list | list.extend([4, 5]) | [1, 2, 3, 4, 5] |
| .insert(i, x) | Inserts x at index i | list.insert(1, 10) | [1, 10, 2, 3] |
| .pop(i) | Removes and returns element at i (default last) | list.pop(1) | [1, 3] |

# Python-learning-Day12

Key Features of Tuples

Data Type & Collection:

Tuples are a collection data type that can store multiple values of different data types (integers, strings, lists, etc.).

Immutable:

Once a tuple is created, its elements cannot be changed, added, or removed.

Indexing:

Tuple elements are accessed using index positions, starting from 0 for the first element.

Tuple Syntax:-

my\_tuple = (1, 2, 3, "Python", 5.5)

single\_tuple = (10,)

Why Use Tuples?

Faster than lists (since they are immutable).

Safe from accidental modifications.

Used as dictionary keys if they contain only immutable elements.

Dictionary in Python

Definition

A dictionary is a built-in data type in Python that stores data in key-value pairs. It is unordered, mutable, and allows for fast lookups.

Key Features of Dictionaries

Key-Value Pairs:

Each element in a dictionary consists of a key and a value.

Syntax: {key: value}

Keys Must Be Unique:

A dictionary key must be unique and immutable (strings, numbers, tuples).

If a duplicate key is used, the latest value will overwrite the previous one.

Values Can Be Unique or Duplicate:

A dictionary value can be duplicated.

Dictionary Syntax:-

# Creating a dictionary

my\_dict = {

"name": "Alice",

"age": 25,

"city": "New York",

"age": 30 # Duplicate key, the last value (30) is stored

}

print(my\_dict)

# Output: {'name': 'Alice', 'age': 30, 'city': 'New York'}

Why Use Dictionaries?

Fast Lookups (compared to lists).

Organized Data Storage (key-value format).

Flexible (supports different data types as values).

Dynamic (can grow and shrink as needed).

# Python-learning-Day13

Set:-

Definition

A set is a built-in data type in Python that stores unordered, unique elements. It is mutable, but its elements must be immutable (e.g., numbers, strings, tuples).

Key Features of Sets

Unordered Collection:

Sets do not maintain the order of elements.

Unique Elements:

Sets do not allow duplicate values.

Mutable (but only the set itself, not its elements):

You can add or remove elements.

Set Syntax:-

# Creating a set

my\_set = {1, 2, 3, 4, 5}

print(my\_set) # Output: {1, 2, 3, 4, 5}

# Creating a set with duplicate values

dup\_set = {1, 2, 2, 3, 4, 4, 5}

print(dup\_set) # Output: {1, 2, 3, 4, 5} (duplicates removed)

# Creating an empty set (must use set(), not {})

empty\_set = set()

print(type(empty\_set)) # Output: <class 'set'>

Why Use Sets?

Fast lookup and membership testing (in operation is faster than in lists).

Automatically removes duplicates from a collection.

Useful for mathematical operations like union and intersection.

# Python-learning-Day 14

1. Valid Parentheses

Problem Statement:

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

A string is valid if:

Open brackets must be closed by the same type of brackets.

Open brackets must be closed in the correct order.

Every close bracket has a corresponding open bracket.

Approach:

Use a stack (Last In, First Out - LIFO).

Push opening brackets onto the stack.

When encountering a closing bracket, check if it matches the top of the stack.

If not, return False.

At the end, the stack should be empty.

Code:

def is\_valid\_parentheses(s):

stack = []

mapping = {')': '(', '}': '{', ']': '['}

for char in s:

if char in mapping: # If it's a closing bracket

top\_element = stack.pop() if stack else '#' # Get top element or dummy value

if mapping[char] != top\_element:

return False

else:

stack.append(char) # Push opening bracket

return not stack # If the stack is empty, return True

# Example usage:

print(is\_valid\_parentheses("()[]{}")) # Output: True

print(is\_valid\_parentheses("(]")) # Output: False

Time Complexity:

O(n) where n is the length of the string (each character is processed once).

2. Pascal's Triangle

Problem Statement:

Given a non-negative integer numRows, generate the first numRows of Pascal's Triangle.

Each number in Pascal's Triangle is the sum of the two numbers directly above it.

Example Output for numRows = 5:

[

[1],

[1,1],

[1,2,1],

[1,3,3,1],

[1,4,6,4,1]

]

Approach:

Use a nested loop:

Start with the first row [1].

Each row starts and ends with 1.

Each middle element is computed as triangle[i-1][j-1] + triangle[i-1][j].

code:

def generate\_pascals\_triangle(numRows):

triangle = []

for i in range(numRows):

row = [1] \* (i + 1) # Create a row with 1s

for j in range(1, i): # Fill in the middle values

row[j] = triangle[i - 1][j - 1] + triangle[i - 1][j]

triangle.append(row)

return triangle

# Example usage:

print(generate\_pascals\_triangle(5))