Python Programming Mind Maps

Developed by Rayees Yaqoob, Sr Lecturer IT, GHSS Kadipora Anantnag

Python Basics & Introduction

Python Programming Language

Easy Syntax
Intuitive syntax that mirrors natural language, accessible for beginners

Interpreted Language
Code executed line by line, easier debugging and rapid development

Dynamic Typing
Variables don't need explicit declaration, automatic memory allocation

Extensive Libraries
Vast standard library and rich ecosystem of third-party packages

Cross-Platform
Runs on Windows, macOS, and Linux operating systems

Strong Community
Large, active community contributing libraries and support

Web Development
- Django
- Flask
- FastAPI

Data Science
- Pandas
- NumPy
- Matplotlib

Machine Learning
- TensorFlow
- Scikit-learn
- PyTorch

Automation

Scripting | Task Automation | System Admin

Getting Started:

1. Install Python from python.org → 2. Choose IDE (PyCharm, VS Code) → 3. Write "Hello, World!" program

Identifiers, Keywords & Variables

Python Naming System

Identifiers

Names used to identify variables, functions, classes, modules, and other objects

```
Valid: user_name, score_1, MAX_SPEED
Invalid: 1stPlace, class, my-variable
```

Keywords

Reserved words with special meaning in Python

```
if, else, for, while, class, import, return, def, try, except, finally
```

Constants

Values that don't change during program execution (uppercase convention)

```
PI = 3.14159
MAX_CONNECTIONS = 100
GRAVITY = 9.8
```

Variables

Used to store data values, can hold different data types

```
age = 25
name = "Alice"
height = 5.6
```

Variable Naming Rules

- ✅ Valid Characters: letters, digits, underscores (_)
- 🚫 Cannot start with a digit
- 🔤 Case-sensitive (age ≠ Age ≠ AGE)
- 🔒 Cannot use reserved keywords
- 📏 No specific length limit (but be descriptive)

💡 Naming Conventions:

snake_case for variables, **UPPER_CASE** for constants, **PascalCase** for classes

⚙️ Python Operators

Python Operators

➕ Arithmetic Operators

```
+ (Addition): 5 + 3 = 8
- (Subtraction): 5 - 3 = 2
* (Multiplication): 5 * 3
= 15
/ (Division): 5 / 2 = 2.5
% (Modulus): 5 % 2 = 1
** (Exponent): 2 ** 3 = 8
```

📝 Assignment Operators

```
= (Assign): x = 5
+= (Add and assign): x +=
3
-= (Subtract and assign):
x -= 2
*= (Multiply and assign):
x *= 4
/= (Divide and assign): x
```

```
// (Floor Division): 5 //
2 = 2
```

```
/= 2
%= (Modulus and assign): x
%= 3
```

🔍 Comparison Operators

```
== (Equal): 5 == 5 → True
!= (Not equal): 5 != 3 →
True
> (Greater than): 5 > 3 →
True
< (Less than): 3 < 5 →
True
>= (Greater or equal): 5
>= 5 → True
<= (Less or equal): 3 <= 5
→ True
```

🧠 Logical Operators

```
and: True and False →
False
or: True or False → True
not: not True → False
```

🆔 Identity Operators

```
is: x is y (same object)
is not: x is not y
(different object)
```

📋 Membership Operators

```
in: 'a' in 'apple' → True
not in: 'z' not in 'apple'
→ True
```

💡 Operator Precedence:

PEMDAS - Parentheses → Exponents → Multiplication/Division → Addition/Subtraction

📊 Python Data Types

Python Data Types

🔢 Numbers

📝 Strings

✅ Boolean

```
int: 42, -17, 0
float: 3.14, -2.5
complex: 3+4j
```

```
"Hello World"
'Python'
"""Multi-line"""
```

```
True
False
```

Lists

```
[1, 2, 3, 4]
['a', 'b', 'c']
[1, 'hello', 3.14]
```

Tuples

```
(1, 2, 3)
('a', 'b', 'c')
(1, 'hello', 3.14)
```

Dictionaries

```
{'name': 'John', 'age': 25, 'city': 'Delhi'}
```

Sets

```
{1, 2, 3, 4}
{'apple', 'banana'}
set([1, 2, 3])
```

Type Checking:

Use **type(variable)** to check data type and **isinstance(variable, type)** to verify type

Indentation, Statements & Expressions

Python Code Structure

Indentation

Defines code blocks using whitespace (4 spaces recommended)

```
if True: print("Indented block") if x > 5: print("Nested indentation")
```

Statements

Instructions that Python interpreter can execute

```
x = 10 # Assignment statement if x > 5: # Control flow statement print("Greater") # Function call import math # Import statement
```

Expressions

Combination of values, variables, and operators that evaluate to a result

```
result = (5 + 3) * 2 # Arithmetic is_greater = (10 > 5) # Relational is_valid = (x > 0) and (x < 100) # Logical length = len("Hello") # Function call
```

Multi-line Statements

```
# Using backslash (\) for continuation total = item_one + \ item_two + \ item_three # Brackets don't need continuation character days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

Indentation Rules:

Use **4 spaces** per level, be **consistent**, avoid mixing spaces and tabs

Input & Output Statements

Python I/O Operations

Input Statements

```
# Basic input (always returns string) name = input("Enter your name: ") # Type casting for numbers age = int(input("Enter age: ")) height = float(input("Enter height: ")) # Multiple inputs x, y = input("Enter two values: ").split()
```

Key Points:

- Always returns string
- Use type casting for numbers
- Can split multiple inputs

Output Statements

```
# Basic output print("Hello, World!") # Multiple values print("Name:", name, "Age:", age) # Formatted output print(f"My name is {name} and I am {age}") # With separators print("A", "B", "C", sep="-")
```

Key Points:

- Can print multiple values
- F-strings for formatting
- Custom separators available

I/O Best Practices:

Always provide clear prompts for input and use meaningful variable names for better code readability

Control Statements

Conditional Statements

IF Statement

IF-ELSE Statement

```
if condition: # Execute if
condition is True
print("Condition is true")
```

Executes code block only if condition evaluates to True

```
if condition: # Execute if
condition is True
print("True block") else: #
Execute if condition is
False print("False block")
```

Provides alternative execution path when condition is False