Important Concepts

Indentation

- Indentation refers to the spaces at the beginning of a code line.
- indentation in Python is very important.
- Python uses indentation to indicate a block of code.
- Python will give an error if we skip the indentation in our code.
- The number of spaces for indentation is up to programmer most common use is four but it has to be at least one.
- we have to use the same number of spaces in the same block of code otherwise Python will give an error.

In [3]:

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

```
Five is greater than two!
Five is greater than two!
```

In [4]:

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

```
  Input In [4]
    print("Five is greater than two!")
    ^
IndentationError: unexpected indent
```

Comments

- in Python # symbol is used for comment.
- Comments which we write in code are ignored by Python.
- Comments can be used to explain Python code.
- Comments are a way to write human-readable documentation for our code and an important part of programming.
- Comments can be placed at the end of a line
- Comments can also be used to prevent execution of some part of code by commenting when testing code.
- Multiline Comment -
  - Python does not really have a syntax for multiline comments.
  - Since Python will ignore string literals that are not assigned to a variable we can use a multiline string for comment.
  - we can add a multiline string using triple quotes in our code

Identifier and Variables

- Identifier - name of variable is called as an identifier.
- Variable - variable is a identifier that denotes the storage.

1. Variables are created when they are first assigned values.
2. Variables are replaced with their values when used in expressions.
3. Variables must be assigned before they can be used in expressions.
4. Variables refer to objects and are never declared ahead of time.

Variable Names

- Rules for Python variables:
  - A variable name must start with a letter or the underscore character.
  - A variable name cannot start with a number.
  - A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
  - Variable names are case-sensitive (age, Age and AGE are three different variables).
  - A variable name cannot be any of the Python keywords.

Naming conversations

- Variable names with more than one word can be difficult to read.
- There are several techniques you can use to make them more readable.
  1. Camel Case :- Each word except the first starts with a capital letter.
     - myVariableName = "John"
  2. Pascal Case :- Each word starts with a capital letter.
     - MyVariableName = "John"
  3. Snake Case :- Each word is separated by an underscore character.
     - my_variable_name = "John"

Types of operators

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Arithmetic Operators

- Arithmetic operators are used with numeric values to perform common mathematical operations.

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y

*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Assignment Operators

- Assignment operators are used to assign values to variables.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison Operators

- Comparison operators are used to compare two values.

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical Operators

- Logical operators are used to combine conditional statements.

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Identity Operators

- Identity operators are used to compare the objects.
- not if they are equal but if they are actually the same object with the same memory location.

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Membership Operators

- Membership operators are used to test if a sequence is presented in an object.

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Bitwise Operators

- Bitwise operators are used to perform operations on binary numbers.

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2

>>	Signed	Shift right by pushing copies of the leftmost bit in	x >> 2

Operator precedence Table

- Operators lower in the table have higher precedence and so bind more tightly in mixed expressions.
- Operators top of the table have less precedence.
- Operators in the same row has same Precedence.so it follows left to right associtivity when combined.

Operators	Description
yield x	Generator function
lambda args: expression	Anonymous function
x if y else z	Ternary selection
x or y	Logical OR (y is evaluated only if x is false)
x and y	Logical AND (y is evaluated only if x is true)
not x	Logical negation
x in y, x not in y	Membership Operation
x is y, x is not y	Object identity tests
x < y, x <= y, x > y, x >= y, x == y, x != y	Magnitude comparison and Value equality operator
x | y	Bitwise OR, set union
x ^ y	Bitwise XOR, set symmetric difference
x & y	Bitwise AND, set intersection
x << y, x >> y	Shift x left or right by y bits
x + y	Addition, concatenation
x – y	Subtraction, set difference
x * y	Multiplication, repetition
x % y	Remainder
x / y, x // y	Division: true and floor
x ** y	Power (exponentiation)
x[i]	Indexing
x[i:j:k]	Slicing
x(...)	function, method or class Call
other	List,Tuple,Dictionary, set, comprehensions

Why we need Operator Precedence?

For Ex. we have a Expression `A * B + C * D` and we want to solve it. So... how does Python know which operation to perform first?

- The answer to this question lies in operator precedence.
- When we write an expression with more than one operator Python groups its parts according to precedence rules and this grouping determines the order in which the expression's parts are computed.

What is the Other Solution ?

- other solution is Parentheses group subexpressions
- we can forget about precedence completely if we're careful to group parts of expressions with parentheses.
- When we enclose subexpressions in parentheses we override Python's precedence rules.
- Python always evaluates expressions in parentheses first before using their results in the enclosing expressions.
    - For Ex. instead of coding X + Y * Z...
        - if we write (X + Y) * Z :- + is performed first then * is performed.
        - if we write X + (Y * Z) :- * is performed first then + is performed.

In [ ]:

```
```