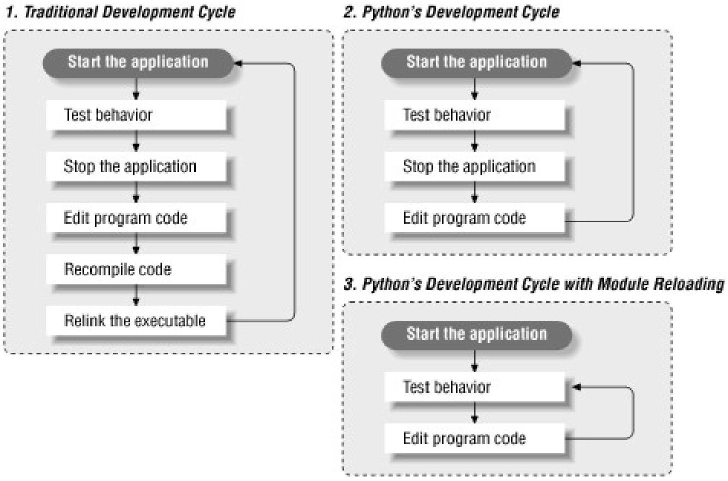
Python

- ✓ Python is a versatile and widely-used programming language known for its readability and ease of use
- ✓ It is object oriented programming language

Programming cycle for Python

- ✓ Python's programming cycle is notably shorter due to its interpreted nature.
- ✓ Python skips compile and link steps, distinguishing itself from traditional languages.
- ✓ Programs import modules at runtime, ensuring immediate execution after changes.
- ✓ Dynamic module reloading in Python allows modifications and reloads during runtime, facilitating continuous program updates without interruptions.



IDE

- ✓ An Integrated Development Environment (IDE) is a software suite that streamlines the software development process.

Top Python IDEs
PyDev
PyCharm
Spyder
Jupyter
Jupyter Notebook
Rodeo
Thonny
Microsoft Visual Studio
eric
Wing
Atom

- ✓ **Key features include:**
  1.Code Editing
  2.Build and Compilation
  3.Project Management
  4.User Interface Design
  5.Extensibility
  6.Documentation and Help
  7.Debugging
  8. Version Control
  9.Testing
  10.Auto-Completion
  11.Performance Profiling

print() Function:

- ✓ The print() function is used to display output on the console.
- ✓ It can take one or more arguments and prints them to the standard output (usually the console).

Syntax

```
print(value1, value2, ..., sep=' ', end='\n')
```

Example :

```
 # Basic print statement
print("Hello, World!")
 # Printing multiple values
name =, nam "Alice"
age = 30
print("Name:"e, "Age:", age)
# Changing separator and end
print("Four")
print("One", "Two", "Three", sep=' | ', end='
*** ')
```

Input function

- ✓ The input() function is used to take user input from the console. It waits for the user to enter some text and returns that text as a string.

Syntax :

```
variable = input(prompt)
```

Example

```
# Taking user input
name = input("Enter your name: ")
print("Hello, " + name + "!")
# Converting input to integer
age = int(input("Enter your age: "))
print("You will be", age + 1, "years old next
year.")
```

Comments

- ✓ **Single-Line Comments:**
  - ➢ Created with #.
  - ➢ Used for short explanations on the same line.
  - ➢ **Syntax** :

    # Single-line comment
- ✓ **Multi-Line (Block) Comments:**
  - ➢ Achieved with triple-quoted strings.
  - ➢ Used for longer explanations spanning multiple lines.
  - ➢ **Syntax** :

```
'''
    Multi-line comment
'''
```

- ✓ **Best Practices:**
  - ➢ Provide clarity and context.
  - ➢ Keep comments concise and up-to-date.
  - ➢ Avoid redundancy with self-explanatory code.
  - ➢ **Example** :

```
result = calculate_total(price, quantity)  # Calculate total cost
```

- ✓ Comments serve to improve code readability and understanding without affecting program execution.

Variables :

- ✓ A variable holds a value that may change.
- ✓ The process of writing the variable name is called declaring the variable.
- ✓ In Python, variables do not need to be declared explicitly in order to reserve memory spaces as in other programming languages like C, Java, etc.
- ✓ When we initialize the variable in Python, Python Interpreter automatically does the declaration process.
- ✓ **Syntax** :

  Variable = Expression
- ✓ **Example** :

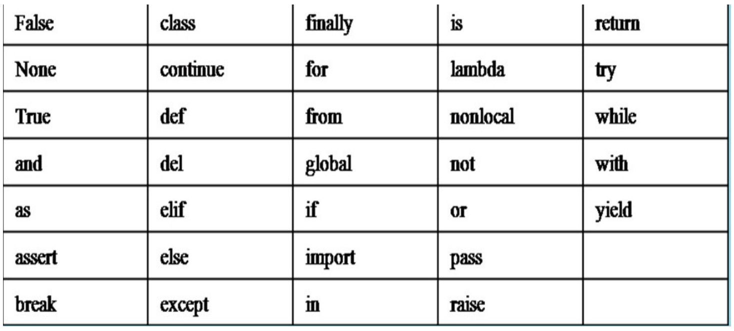
  x = 10 # Variable x is initialized with the value 10

Identifier

- An identifier is a name given to a variable, function, class, module, or other entities in a program.
- **Rules for Naming:**
  - Must start with a letter (a-z, A-Z) or an underscore (_).
  - The remaining characters can be letters, numbers (0-9), or underscores.
  - Case-sensitive (e.g., variable and Variable are different).
  - Cannot be a reserved word (e.g., if, else, for, etc.).

Reserve keywords

- Reserved keywords in a programming language are words that have special meanings and cannot be used as identifiers (names for variables, functions, etc.)
- because they are reserved for specific purposes in the language.



False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Data types

- data types represent the type of data that a variable can hold.
- Here are some common data types in Python:
- **Integer (int):**
  - Represents whole numbers without decimal points.
  - **Example**: x = 5
- **Float (float):**
  - Represents numbers with decimal points or in exponential form.
  - **Example**: y = 3.14
- **String (str):**
  - Represents text or sequences of characters.
  - **Example**: name = "John"
- **Boolean (bool):**
  - Represents either True or False.
  - **Example**: is_valid = True
- **List (list):**
  - Represents an ordered collection of elements.
  - **Example**: numbers = [1, 2, 3]
- **Tuple (tuple):**
  - Similar to a list but immutable (cannot be changed after creation).
  - **Example**: coordinates = (4, 5)
- **Set (set):**
  - Represents an unordered collection of unique elements.
  - **Example**: unique_numbers = {1, 2, 3}
- **Dictionary (dict):**
  - Represents a collection of key-value pairs.
  - **Example**: person = {'name': 'Alice', 'age': 25}
- **NoneType (None):**
  - Represents the absence of a value or a null value.
  - **Example**: result = None
- **Complex (complex):**
  - Represents complex numbers with a real and an imaginary part.
  - **Example**: z = 3 + 4j

Type conversion

- Type conversion in Python is changing the data type of a variable, allowing operations involving different data types.

Implicit Conversion:

- Automatic conversion by Python based on the operation being performed.

Explicit Conversion (Casting):

- Use specific functions for explicit conversion:
  - int() for integer conversion.
  - float() for float conversion.
  - str() for string conversion.

Examples:

```
# Implicit conversion
result = 10 + 3.14  # int implicitly converted to float for addition
# Explicit conversion
num_float = 5.5
num_int = int(num_float)  # Converts float to integer
str_num = str(42)
```

Input Function Note:

- input() returns a string, so use int() or float() for numerical input.
  - age_str = input("Enter your age: ")
  - age_int = int(age_str) # Converts input string to integer

Expression:

- A combination of symbols that yields a value.
- Comprises variables, operators, values, sub-expressions, and reserved keywords.
- When entered in the command line, expressions are evaluated by the interpreter, producing a result.
- **arithmetic expressions are** evaluating to a numeric type are termed arithmetic expressions.
- **Sub-expressions** are parts of larger expressions, enclosed in parentheses.
- **Example**: 4 + (3 * k)
- **Individual Expressions:** A single literal or variable can also be considered an expression (e.g., 4, 3, k).
- **Hierarchy of Sub-Expressions:** Expressions may have multiple levels of sub-expressions, forming a hierarchy.
- **Example**: a + (b * (c - d)) has sub-expressions 3 and k.

Purpose of Assignment Statements:

- Used for creating, assigning values to, and modifying variables.
- **Syntax**:
- Variable = Expression represents the assignment statement.

Types of Assignment Statements:

- **Value-based expression on RHS.**
  x = 5 + 3 # Assign the result of the expression (5 + 3) to the variable x
- **Current variable on RHS.**
  y = 2
  y = y + 1 # Increment the value of the variable y by 1
- **Operation on RHS.**
  a = 10
  b = 2
  c = a / b # Assign the result of the division operation (a / b) to the variable c

operators

- operators are special symbols or keywords that perform operations on operands.

Types of operators

Arithmetic Operators:

- ✓ Perform basic mathematical operations.
- ✓ Examples: + , - , * , / , % , **

Program:

```
a=5
b=10+a
Print(a, b ) #output  5 15
```

Comparison Operators:

- ✓ Compare two values and return a Boolean result.
- ✓ Examples: == , != , < , > , <= , >=

Program

```
a=5
b=10
Print(a==b , a> b , a< b  ) #output:
false  false  true
```

Logical Operators:

- ✓ Perform logical operations on Boolean values.
- ✓ Examples: and , or, not

Program

```
a=5
b=10
Print( a> b and  a< b  ) #output: false
Print( a> b or a< b  ) #output: true
Print( not  a< b  ) #output: false
```

Assignment Operators:

- ✓ Assign values to variables.
- ✓ Examples: =, +=, -= , *= , /=

Program

```
a=5
a+=6
Print(a   ) #output: 11
```

Bitwise Operators:

- ✓ Perform operations on individual bits of binary numbers.
- ✓ Examples: & , | , ^ , ~ , << , >>

Program

```
a=5
b=6
Print(a & b , ~b   ) #output:  4, -7
```

Membership Operators:

- ✓ Check if a value is a member of a sequence.
- ✓ Examples: in , not in

Program

```
a=[5.,3,2,3]
b=4
Print(b in a , b not in a ) #output: false   true
```

Identity Operators:

- ✓ Compare the memory location of two objects.
- ✓ Examples: is ,is not

Program

```
 a=5
b=a
Print(a is b , a is not b   ) #output: true false
```

Operator precedence

Operators	Associativity
() Highest precedence	Left - Right
**	Right - Left
+x , -x, ~x	Left - Right
*, /, //, %	Left - Right
+, -	Left - Right
<<, >>	Left - Right
&	Left - Right
^	Left - Right
|	Left - Right
Is, is not, in, not in, <, <=, >, >=, ==, !=	Left - Right
Not x	Left - Right
And	Left - Right
Or	Left - Right
If else	Left - Right
Lambda	Left - Right
=, +=, -=, *=, /= Lowest Precedence	Right - Left

- ✓ Operator precedence defines the order in which different operators are evaluated in an expression. Operators with higher precedence are evaluated first.

Associativity :

- ✓ Determines the order of execution for operators with the same precedence.

Two Types of Associativity:

a. Left to Right:

- ✓ Operators of the same precedence are executed from the left side first.

b. Right to Left:

- ✓ Operators of the same precedence are executed from the right side first.

Associativity in Python:

- ✓ Left-to-right associative operators include multiplication, floor division, etc.
- ✓ The ** operator is right-to-left associative.

Example

- ✓ result_left = 5 + 3 - 2 # Addition and subtraction with left-to-right associativity
- ✓ print(result_left) # Output: 6
- ✓ result_right = 2 ** 3 ** 2 # Exponentiation with right-to-left associativity

print(result_right) # Output: 512

PEP 8

- ✓ PEP 8 is the Python Enhancement Proposal that provides style guidelines for writing clean, readable, and consistent Python code.
- ✓ Here are some key points from PEP 8
- ✓ **Indentation:**Use 4 spaces per indentation level.
- ✓ **Maximum Line Length:**Limit all lines to a maximum of 79 characters for code, and 72 for docstrings.
- ✓ **Imports:**Import standard libraries first, followed by third-party libraries, and then your own modules.
- ✓ **Whitespace in Expressions and Statements:**
  - ➢ Immediately inside parentheses, brackets, or braces.
  - ➢ Immediately before a comma, semicolon, or colon.
- ✓ **Comments:**Comments should be used to explain complex pieces of code or decisions that may not be obvious.
- ✓ **Naming Conventions:**
  - ➢ Use snake_case for function and variable names.
  - ➢ Use CamelCase for class names.
  - ➢ Avoid single-letter variable names except for indices and loop variables.
- ✓ **Docstring Conventions:**
  - ➢ Use triple double-quotes for docstrings.
  - ➢ Write docstrings for all public modules, functions, classes, and methods.

1. **Write a Python program that declares three variables (integer, string, and float) and prints their values**
2. **Calculate the average of three numbers**
3. **Create a program that takes two numbers as input and calculates their sum, difference, product, and quotient.**

4. **Write a program that compares two numbers and prints whether they are equal, greater, or smaller.**

5. **Develop a program that takes two strings as input and concatenates them. Print the resulting string.**

6. **Implement a Python program that demonstrates incrementing a variable by 1 and then decrementing it by 1.**

7. **Create a Python program that swaps the values of two variables without using a temporary variable.**

8. **Develop a program that uses compound assignment operators to update the values of variables.**

9. **Write a Python program that takes user input for their name, age, and favorite color. Print a formatted output using this information.**