PYTHON

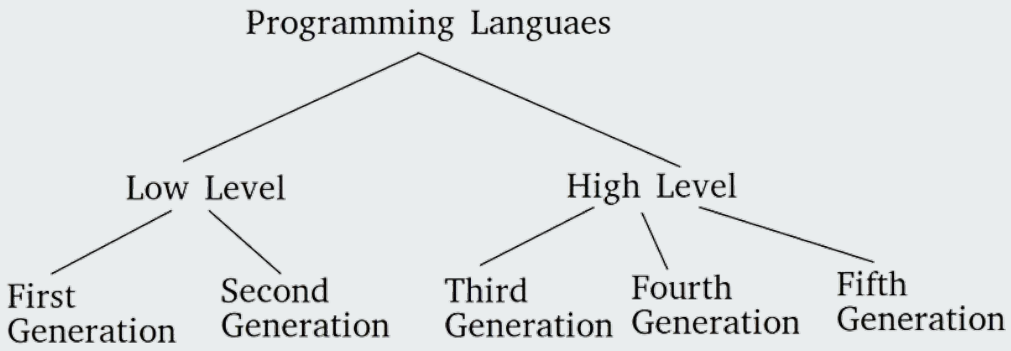
MANAS RAHMAN

Introduction to Computer Program

- Algorithm:
    - From programming point of view, an algorithm is a step-by-step procedure to resolve any problem.
    - An algorithm is an effective method expressed as a finite set of well-defined instructions.
    - Can write using Natural languages.
    - Properties:
        - Input
        - Output
        - Definiteness: Every statement of the algorithm should be unambiguous.
        - Effectiveness: It should be effective to solve the problem.
        - Finiteness: It should be finite (no infinite loops).
- Program: A computer program is a **sequence of instructions** written using a **Computer Programming Language** to perform a specified task by the computer.
- Programming: The act of writing computer programs is called computer programming.
- Uses of Computer Programs: Today computer programs are being used in almost every field, household, agriculture, medical, entertainment, defense, communication, etc.
- Applications of computer programs: MS Word, MS Excel, Adobe Photoshop, Internet Explorer, Chrome, etc.

Programming Language

- ❑ First-generation: Machine language
- ❑ Second-generation: Assembly language
- ❑ Third-generation: High-level language
- ❑ Fourth-generation: Database programming and scripting
- ❑ Fifth-generation: Visual tools



First-generation: Machine language

- ❑ A set of primitive instructions built into every computer
- ❑ The instructions are in the form of binary code
  - ❑ 1101101010011010
  - ❑ 3 is represented as: 011

Advantages :

1. Fast & efficient as statements are directly written in binary language.
2. No translator is required.

Disadvantages :

1. Difficult to learn binary codes.
2. Difficult to understand – both programs & where the error occurred.

Second-generation: Assembly language

❑ Contains human-readable notations that can be further converted to machine language using an assembler

Eg: ADD R1, R2, R3

❑ Combination of mnemonic(symbolic name for a single executable machine language instruction, an opcode) and machine instruction

❑ Used in kernels and hardware drives

Advantages :

1. It is easier to understand if compared to machine language.
2. Modifications are easy.
3. Correction & location of errors are easy.

Disadvantages :

1. Assembler is required.
2. This language is architecture /machine-dependent, with a different instruction set for different machines.

Third-generation: High-level language

- ❑ English-like words that humans can understand easily.

  Eg: Area = 5 * 5 * 3.1415;

- ❑ For execution, a program in this language needs to be translated into machine language using a Compiler/ Interpreter.
- ❑ Examples of this type of language are C, C++, C#, JAVA, PASCAL, FORTRAN, COBOL, etc.

Advantages :

1. Use of English-like words makes it a human-understandable language.
2. Lesser number of lines of code as compared to the above 2 languages.
3. Same code can be copied to another machine & executed on that machine by using compiler-specific to that machine.

Disadvantages :

1. Compiler/ interpreter is needed.
2. Different compilers are needed for different machines.

Fourth-generation: Database programming and scripting

- ❑ Consist of statements that are similar to statements in the human language.
- ❑ These are used mainly in database programming and scripting.
- ❑ Examples of these languages include Perl, Python, Ruby, SQL, and MatLab(MatrixLaboratory).

Advantages :

1. Easy to understand & learn.
2. Less time is required for application creation.
3. It is less prone to errors.

Disadvantages :

1. Memory consumption is high.
2. Has poor control over Hardware.
3. Less flexible.

Fifth-generation: Visual tools

- ❑ Programming languages that have visual tools to develop a program.
- ❑ Examples of fifth-generation languages include Mercury, OPS5, and Prolog.
- ❑ It uses the concept that rather than solving a problem algorithmically, an application can be built to solve it based on some constraints, i.e., we make computers learn to solve any problem.

Advantages :

1. Machines can make decisions.
2. Programmer effort reduces to solve a problem.
3. Easier than 3GL or 4GL to learn and use.

Disadvantages :

1. Complex and long code.
2. More resources are required & they are expensive too.

Introduction to Python

- **Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.**
- Created by Guido van Rossum during 1985- 1990.
- Got the name after “Monty Python's Flying Circus”, a BBC comedy series from the 1970s.
- Van Rossum thought he needed a name that was short, unique, and slightly mysterious, so he decided to call the language Python.
- Source code available under the GNU General Public License (GPL).

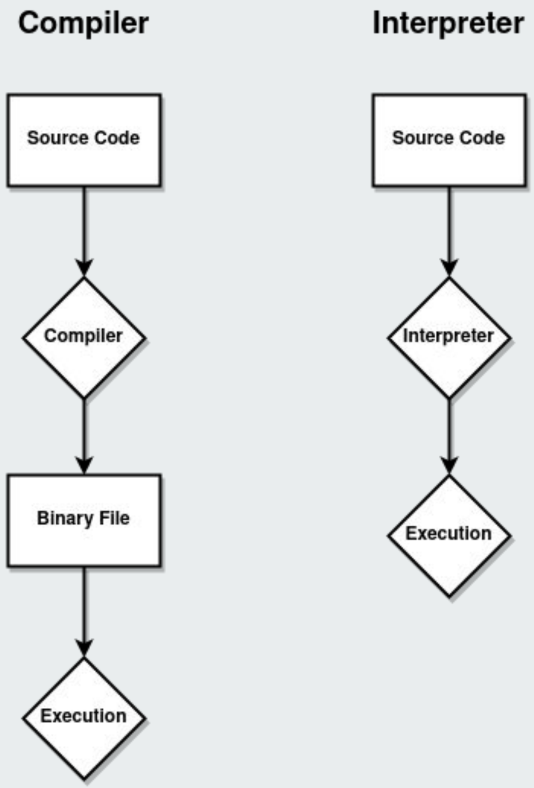
Introduction to Python (Cont…)

- ❑ General-purpose: for professional web development also.
- ❑ Interpreted: You do not need to compile your program before executing it.
- ❑ Interactive: You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- ❑ Object-oriented: Programming that encapsulates code within objects.
- ❑ High-level programming language.
- ❑ Wide range of applications from simple text processing to WWW browsers to games.

Features of Python

- ❑ Python programs is easily portable.ie it works in almost all platforms
- ❑ Python uses in internet scripting, robotic programming, machine learning etc
- ❑ Python contains rich set of libraries.
- ❑ Python is case-sensitive. For example, NAME and name are not same in Python.
- ❑ Python is portable and platform independent, means it can run on various operating systems and hardware platforms.
- ❑ Python has a rich library of predefined functions.
- ❑ Python is also helpful in web development. Many popular web services and applications are built using Python.
- ❑ Python uses indentation for blocks and nested blocks.

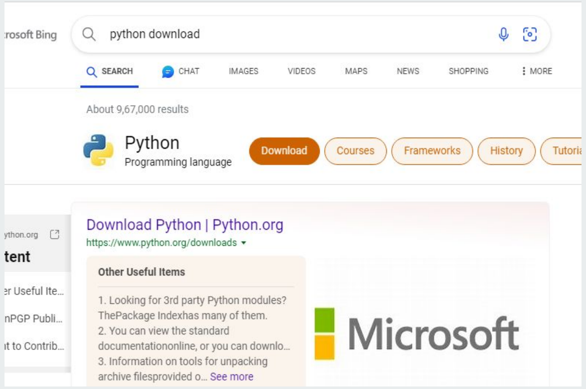
Working of Compiler and Interpreter



Difference between Compiler and Interpreter

Compiler	Interpreter
Takes Entire program as input	Takes Single instruction as input
Intermediate Object Code is Generated	No Intermediate Object Code is Generated
Conditional Control Statements are Executes faster	Conditional Control Statements are Executes slower
Memory Requirement is More(Since Object Code is Generated)	Memory Requirement is Less
Program need not be compiled every time	Every time higher level program converted into lower level program
Errors are displayed after entire program is checked	Errors are displayed for every instruction interpreted
Example : C Compiler	Example : PYTHON

Installation



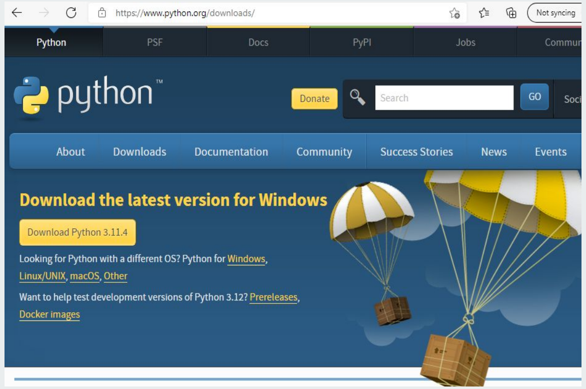
About 9,67,000 results

Python
Programming language

Download Python | Python.org
https://www.python.org/downloads

Other Useful Items

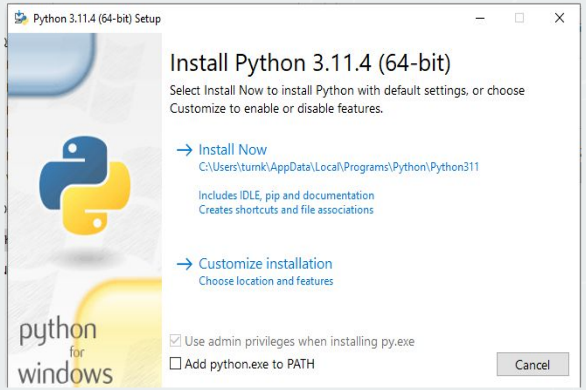
1. Looking for 3rd party Python modules? ThePackage Indexhas many of them.
2. You can view the standard documentationonline, or you can downlo...
3. Information on tools for unpacking archive filesprovided o...



Download the latest version for Windows

Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other

Want to help test development versions of Python 3.12? Prereleases, Docker images



Install Python 3.11.4 (64-bit)

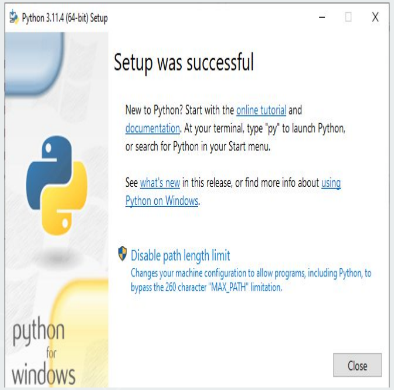
Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

→ Install Now
C:\Users\turnk\AppData\Local\Programs\Python\Python311

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

[✓] Use admin privileges when installing py.exe
[ ] Add python.exe to PATH

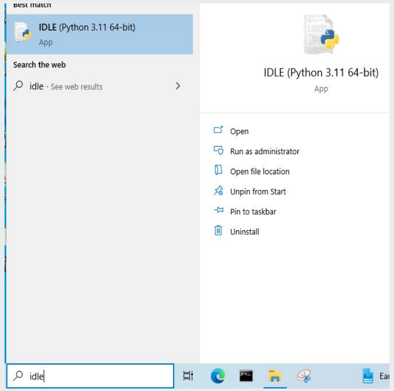


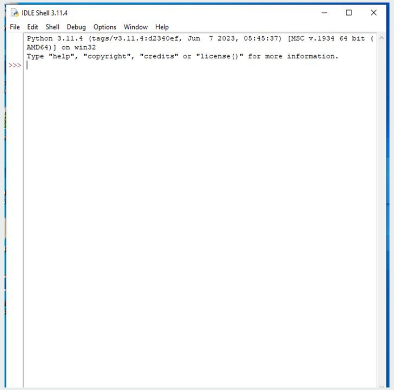
Setup was successful

New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

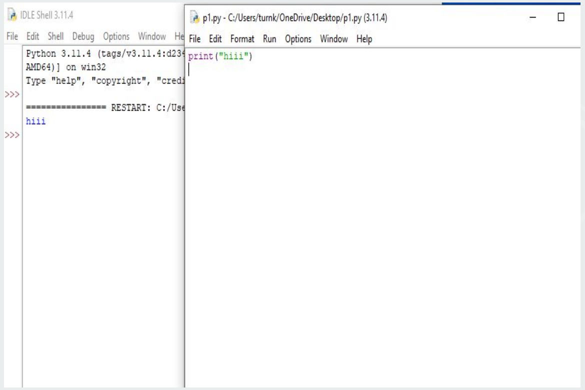
See what's new in this release, or find more info about using Python on Windows.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.





```
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```



```
Python 3.11.4 (tags/v3.11.4:d234
AMD64)] on win32
Type "help", "copyright", "credi
>>>
================ RESTART: C:/Use
hiii
>>>
```

```
print("hiii")
```

MANAS RAHMAN

Modes of python interpreter

1. Interactive mode

Interactive Mode, as the name suggests, allows us to interact with OS.

When we type Python statement, interpreter displays the result(s) immediately.

Advantages:

Python, in interactive mode, is good enough to learn, experiment or explore.

Working in interactive mode is convenient for beginners and for testing small pieces of code.

Drawback:

We cannot save the statements and have to retype all the statements once again to re-run them.

In interactive mode, you type Python programs and the interpreter displays the result:

```
>>> 1 + 1
2
>>> print ('Hello, World!')
Hello, World!
```

Modes of python interpreter (Cont…)

2. Script mode

In script mode, we type python program in a file and then use interpreter to execute the content of the file.

Scripts can be saved to disk for future use.

Python scripts have the extension .py

Save the code and run the interpreter in script mode to execute the script.

Comments

- ❑ Comments are used to add a remark in the source code.
- ❑ Comments are not executed by the interpreter.
- ❑ Comments provide readability and understandability of the program code.
- ❑ By providing comments , it is very easy to maintain the source code
- ❑ Python uses the hash character(#) for single line comments and 3 quotes(‘’’) for multiline comments.

Eg:

```
# program to add two numbers
a = 10
b = 20
c =  a+b          # addition operation
print(‘result is ‘,c)
```

output

```
          result is 30
```

Identifier

- Python identifier is a name used to identify a variable , functions, class, module or other objects.
- Rule for creating identifiers:
  - The name should begin with an uppercase or a lowercase alphabet or an underscore sign (_).
  - This may be followed by any combination of characters, alphabets, 0–9 or underscore (_).
  - It can be of any length.
  - It should not be a keyword.
  - Do not use special characters
  - Do not use white spaces.

Keyword

- ❑ Keyword is reserved word, it has a specific meaning and purpose.
- ❑ Its meaning cannot be changed.
- ❑ All keywords are written in lowercase letters except True, False and None keywords.
- ❑ Some python keywords are:

and	del	from	elif
if	else	while	in
True	class	for	False

Variables

- ❑ A variable is an identifier and A container (storage area) to hold data.
- ❑ A variable is a name given to store or hold a data value .
- ❑ A variable may change during the execution of a program.
- ❑ Its meaning cannot be changed.
- ❑ In python variable need not be declared explicitly , it will be automatically declared at the time of initialization.
- ❑ Assigning values to Variables (using =)

```
number = 10
```

- ❑ Changing the Value of a Variable:

```
N = 10
N = 20
```

- ❑ Assigning multiple values to multiple variables:

```
a, b, c = 10, 12, 13
```

Constants and User Input

- ❑ Constants:
  - ❑ A constant is a special type of variable whose value cannot be changed.
  - ❑ Use the syntax as the variables.
  - ❑ In Python, constants are usually declared and assigned in a module (a new file containing variables, functions, etc which is imported to the main file).
- ❑ User Input:
  - ❑ Python allows for user input.
  - ❑ That means we are able to ask the user for input.
  - ❑ The method is a bit different in Python 3.6 than Python 2.7.
  - ❑ Python 3.6 uses the input() method: takes the input from the user and evaluates the expression, which means Python automatically identifies whether user entered a string or a number or list.

    ```
    input("<message>")
    ```
  - ❑ Python 2.7 uses the raw_input() method:takes exactly what is typed from the keyboard, convert it to string and store in our variable.

Data Types

- ❑ Specifies the type of data used in the program
- ❑ Python has three categories of data types
    - ❑ Basic types : integer or int, float,complex,bool, string
    - ❑ Container types: list, tuple, set, dict
    - ❑ User-defined types : class

Basic types: Integer or int

- ❑ This value is represented by int class.
- ❑ Contains positive or negative whole numbers (without fraction or decimal).
- ❑ There is no limit on size of the number.

Eg:

```
>>>num=345            # integer number
```

Basic types: float

- ❑ This value is represented by float class.
- ❑ It is a real number with floating point representation.
- ❑ Specified by a decimal point.
- ❑ Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.

  Eg:

```
>>>height=2.5          # real(float) number
```

Basic types: complex

- ❑ Complex number is represented by complex class.
- ❑ A complex number has real and imaginary part.
- ❑ We can use letter “j” to represent the imaginary part.

  Eg:

```
>>>root=2+5j                # complex number
```

Basic types: bool

- Data type with one of the two built-in values, True or False.

  Eg:

  ```
  >>>a = True        #True means 1
  ```

Basic types: String

- ❑ Data String is a group of characters. These characters may be alphabets, digits or special characters including spaces.
- ❑ String values are enclosed either in single quotation marks (e.g., ‘Hai’) or in double quotation marks (e.g., “Hai”)
- ❑ Many operations can be performed on string , they are A list can use the operator plus[+] for concatenation, asterisk[*] for repetition and slicing[:] for sublist.

  Eg:

```
first = "Sherin"        second = "Sam"        print(first+second)        print(first)
first=first+"Riza"      print(first)          print(first*3)             print(first)
print(first[0:3])       print(first[-4:])     print(first[-1])
```

Container type

- ❑ Container types contains multiple values stored together. They include list, tuple, set, dict.
- ❑ The following terms are used for describing container types.
    - ❑ Collection: group of elements
    - ❑ Iterable: elements can be iterated using loop
    - ❑ Ordered collection: element are stored the same order in which they are inserted. So element can be accessed using index or the position in the collection
    - ❑ Unordered collection: elements not stored in the same order in which they are inserted. so we cannot predict the order of an element using index.
    - ❑ Sequence: is the generic term for ordered collection
    - ❑ Immutable : element cannot be changed once inserted
    - ❑ Mutable : element can be changed

Container type: List

- List is a most used data type for storing collection of elements in python.
- A list can contain same type or different types of elements.
- Allows duplicates.
- A list is ordered and indexable.
- List is mutable and iterable.
- List is a sequence of items separated by commas and the items are enclosed in square brackets [ ].
- A list can use the operator plus[+] for concatenation, asterisk[*] for repetition and slicing, [:] for sublist

Eg:

```
first = [1,"Don",5.6,"Riya"]        second=["Jose","Shekar"]        print(first)
print(second)                       print(first+second)             print(first*3)
print(first[1])                     print(first[0:2])               print(first[-1])
```

Container type: Tuple

- ❑ Tuple is a sequence of items separated by commas and items are enclosed in parenthesis ( ).
- ❑ Tuple is almost similar to list. Tuple is read only list.
- ❑ A tuple is ordered and index able.
- ❑ Tuple is immutable and iterable.
- ❑ A list can use the operator plus[+] for concatenation, asterisk[*] for repetition and slicing[:] for sublist

Eg:

```
first = (1,"Don",5.6,"Riya")      second=("Jose","Shekar")      print(first)
print(second)                     print(first+second)           print(first*3)
print(first[1])                   print(first[0:2])             print(first[-1])
```

Difference between list and tuple

list	tuple
Used to represent sequence of item	Used to represent sequence of item
Allows similar and different data elements	Allows similar and different data elements
Elements are enclosed in square brackets[ ]	Elements are enclosed in parenthesis( )
List is mutable ie once element is stored then can be modified first[0] = 10 #available	Tuple is immutable ie once element is stored then cannot be modified first[0] = 10 #not available

Container: Set

- Set is an unordered collection of items separated by commas and the items are enclosed in curly brackets { }.
- It cannot have duplicate entries.
- Once created, elements of a set cannot be changed(immutable).
- In this cannot access the element using index. It can contain same or different types of data.
- As in the set operation in mathematics , python also provide the operator like Union(|), intersection (&), difference(-) and symmetric difference(^) be applied on set.

  Eg:

```
set1={1,2,3,4,5}        set2={4,5,6,7}        print(set1)           print(set2)
uni=set1 | set2         print(uni)            inter=set1 & set2     print(inter)
dif=set1 - set2         print(dif)            adif=set1 ^ set2      print(adif)
```

Container: Dictionary

- ❑ As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered
- ❑ Items in a dictionary are enclosed in curly brackets { }.
- ❑ Dictionaries permit faster access to data.
- ❑ Every key is separated from its value using a colon (:) sign.
- ❑ The key : value pairs of a dictionary can be accessed using the key.
- ❑ The keys and their values can be any data type.
- ❑ In order to access any value in the dictionary, we have to specify its key in square brackets [ ].

Eg:

```
dic={1:"econometrics",2:"computer",3:"maths"}          dic[4]="statistics"
print(dic)        print(dic.keys())        print(dic.values())        print(dic[4])
```

User-defined type: Classes and objects

- In python every type is a class. So int, float , bool , list … all are classes. These are ready made or built in class.
- **Python also has user defined class.**
- A class contains variable and methods.
- An object is created from a class. It is an instance of a class.
- Multiple objects can be created from a class.
- A class has a name and object is nameless.
- Object address of memory can be obtained using id() function.

  Eg:

```
a=12        b="TheSample"        print(a,b)        print(type(a),type(b))
print(id(a),id(b))        print(isinstance(a,int),isinstance(b,str))
```

Operators and Operands

- ❑ Operator is a symbol in which used to operate on operand.
- ❑ Operand means data or variable used in the expression
- ❑ Eg: b= a + 10 the operator is +(addition) and operands are a ,b and 10
- ❑ Different operator in python are classified as follows:
    - ❑ Arithmetic operators
    - ❑ Comparison operators
    - ❑ Logical Operators
    - ❑ Bitwise Operators
    - ❑ Assignment Operators
    - ❑ Membership Operators
    - ❑ Identity Operators

Arithmetic operators

- Arithmetic operators are used to perform arithmetic operations between two operands.
- It includes + (addition), - (subtraction), *(multiplication), /(divide), %(reminder), //(floor division), and exponent (**) operators.

  Eg

```
a = 10          b = 12
print(a + b)                    # 22
print(a - b)                    #-2
print(a * b)                    #120
print(b / a)                    #1.2
print(b % a)                    #2
print(5 ** 2)                   #25
print(b // a )                  #1
```

Comparison Operator

- ❏ Comparison operator also known as relational operator.
- ❏ Which is used to compare values.
- ❏ Result of this operation is always Boolean values either true or false.

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

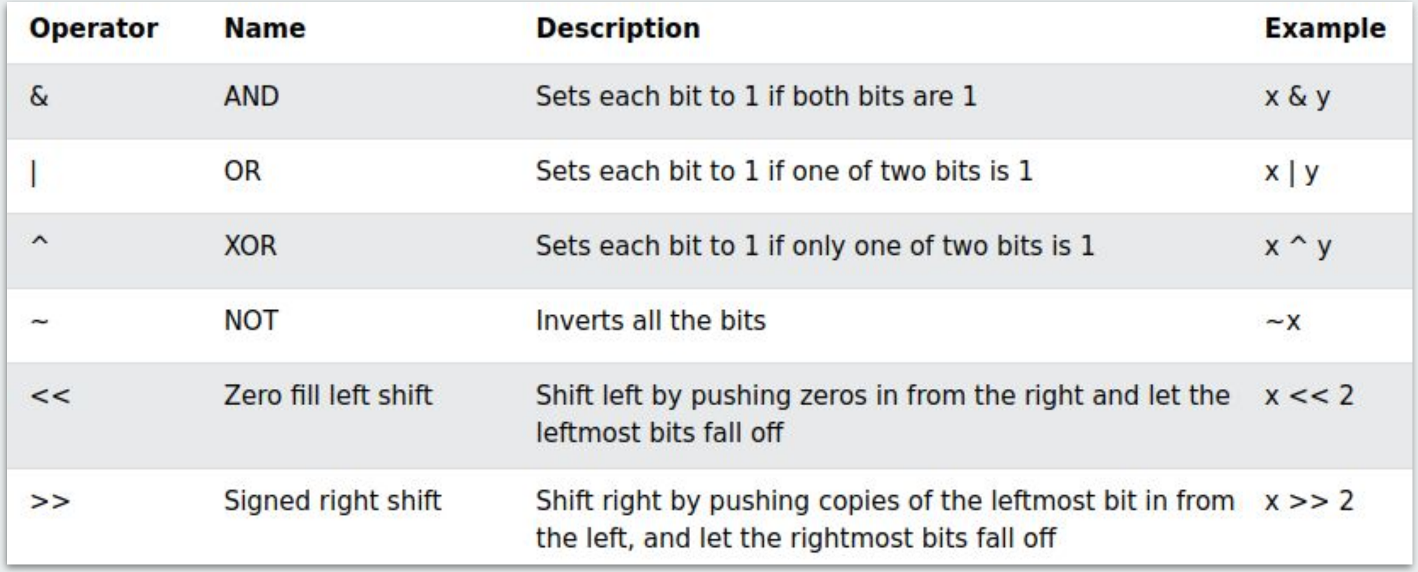
Logical Operators

- ❑ These operators are used to check two or more conditions.
- ❑ The result of these operand is always either true or false Boolean values.

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)

Bitwise Operators

- These operator perform bit level operation on operands.
- Let us take two operands x=10 and y=12.
- In binary format can be written as x=1010 and y=1100.



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 right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

Bitwise Operators (Cont…)

- NOT:

$x = (10)_{10} = (1010)_2 = (\underline{0}1010)_2$ #including sign bit

$\sim x = (\underline{1}0101)_2$ ✖ #can’t be stored as it is in computer (need 2’)

$\sim x = (\underline{1}1011)_2 = -11$ #same as 2’ of x and $\sim x = -(x+1)$

For negative numbers:

$x = (-10)_{10}$ #will be stored as 2’s complement form only

$(10)_{10} = (1010)_2 = (\underline{0}1010)_2$ #+ve number is like this (10)

$\Rightarrow (-10)_{10} \Rightarrow (\underline{1}0101)_2 \Rightarrow (\underline{1}0110)_2$ #2’s complement of +ve number (-10)

$x = (\underline{1}0110)_2$ # including sign bit

$\sim x = (\underline{0}1001)_2 = (9)_{10}$ #~x = -(x+1)

Bitwise Operators (Cont…)

- Left Shift:

  $x = 10 = (1010)_2$

  $x << 1 \Rightarrow (10100)_2 = 20$ #double of x $(x << 2 \Rightarrow (101000)_2 = 40)$

  | 1 | 0 | 1 | 0 |

  | 1 | 0 | 1 | 0 | 0 | #depend on bit size of system

- Right Shift:

  $x = 10 = (1010)_2$

  $x >> 1 \Rightarrow (0101)_2 = 5$ #half of x $(x >> 2 \Rightarrow (0010)_2 = 2)$

  | 1 | 0 | 1 | 0 |

  | 0 | 1 | 0 | 1 | #last digit will be removed

Assignment Operators

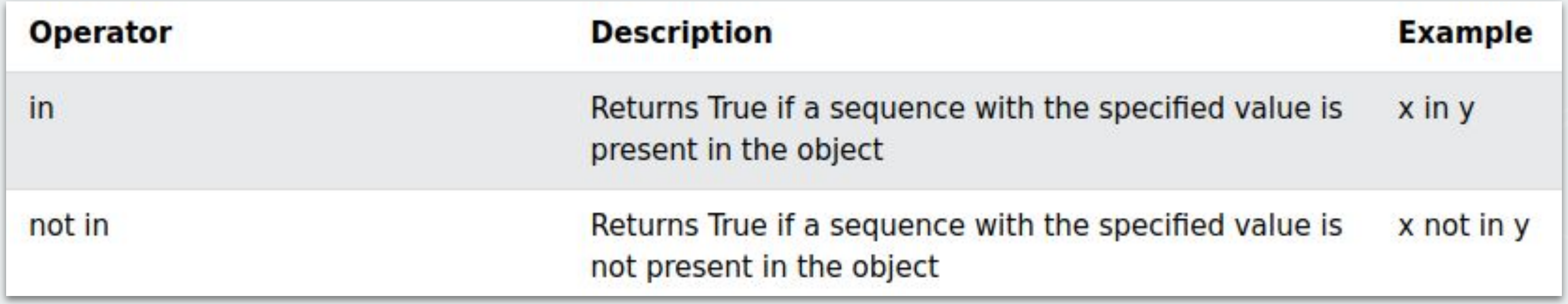
- The operator is used to store right side operand in the left side operand.

Operator	Example
=	x = 5
+=	x += 3
-=	x -= 3
*=	x *= 3
/=	x /= 3
%=	x %= 3
//=	x //= 3

Operator	Example	
**=	x **= 3	(power)
&=	x &= 3	(bitwise AND)
|=	x |= 3	(in-place operation, set: union, num: bitwise OR)
^=	x ^= 3	(bitwise XOR, both same = 0 else = 1)

Membership Operators

- ❑ These operators are used to check an item or an element that is a part of string , a list or a tuple.
- ❑ A membership operator reduces the effort of searching an element in the list.
- ❑ Suppose x stores the value 20 and y is the list containing item 10,20,30 and 40. Then x is a part of the list y because the value 20 is in the list y.

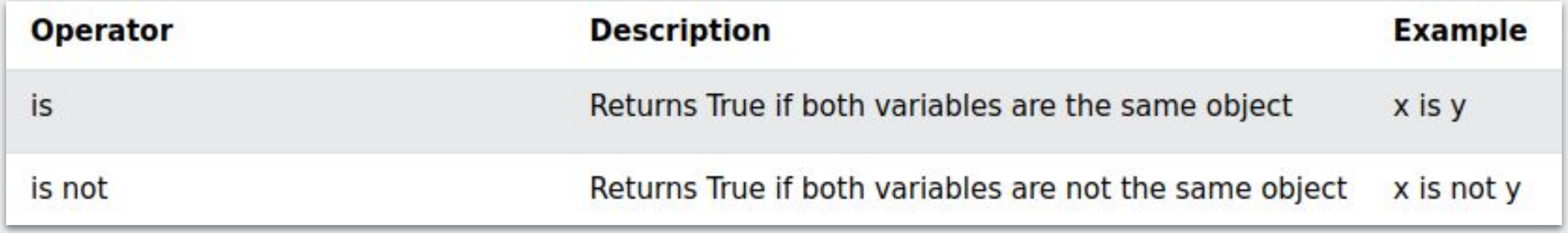


Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Identity Operators

- These operator are used to check whether both operands are same or not.
- Suppose x stores a value 20 and y stores a value 40.

  Then x is y returns false and x is not y returns true



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

Eg:

```
x = [1, 2]      y = [1, 2]      z = x
print(x is z)                           #true (x is same as z)
print(x is y)                           #false (x is not same object as y)
print(x == y)                           #true (x is equal to y)
```

Precedence of operator

- When the expression has two or more operators we need to identify the correct sequence to evaluate these operators.
- This because the final answer changes depending on the sequence thus chosen.

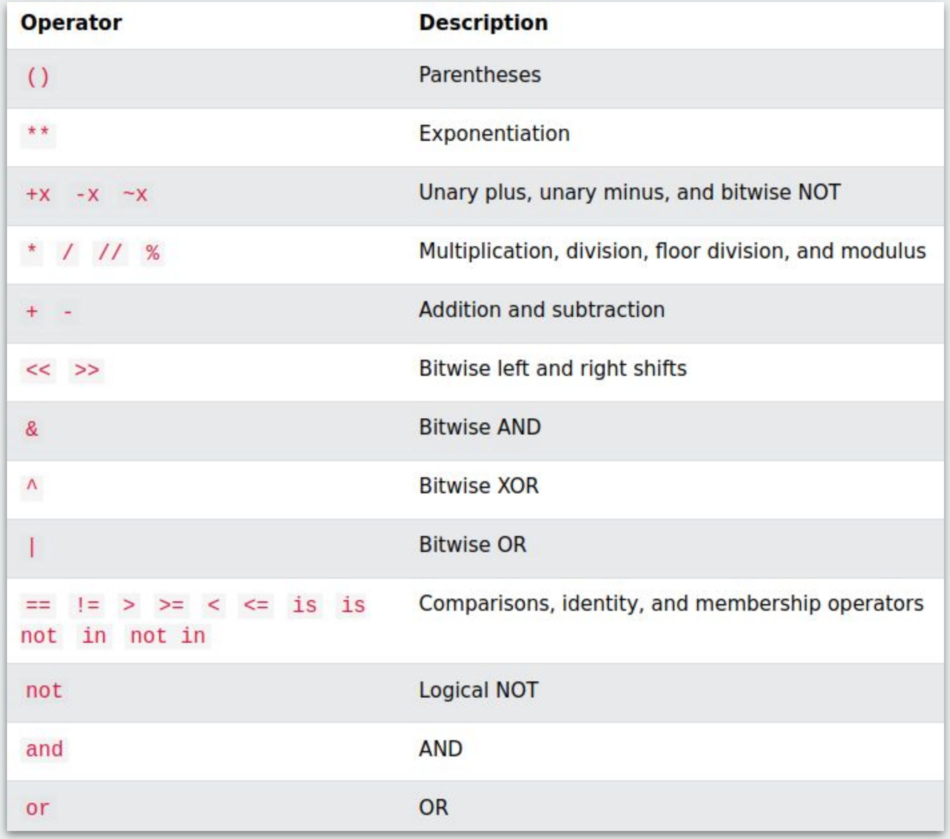
  Example: 10 + 5 / 5

  When the given expression is evaluated from left to right then the final answer is 3.
- Precedence means the order in which the operator is evaluated in a expression which contain many different operators.
- All the operators except Exponentiation(**) operator has a left to right precedence while an equation contains operators with same precedence.

  Example: 2**2**3

  The given expression is evaluated as: 2*2 * 2*2 * 2*2 * 2*2 = 256

Precedence of operator (Cont…)



Operator	Description
`()`	Parentheses
`**`	Exponentiation
`+x` `-x` `~x`	Unary plus, unary minus, and bitwise NOT
`*` `/` `//` `%`	Multiplication, division, floor division, and modulus
`+` `-`	Addition and subtraction
`<<` `>>`	Bitwise left and right shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`==` `!=` `>` `>=` `<` `<=` `is` `is not` `in` `not in`	Comparisons, identity, and membership operators
`not`	Logical NOT
`and`	AND
`or`	OR

Conditional Statements

❑ If	
❑ If …. else &	elif
❑ Shorthand if	if <condition>: <single_line_statement>
❑ Shorthand if ... else	<true_statement> if <condition> else <false_statement>
❑ And	if <condition_1> and <condition_2>:
❑ Or	if <condition_1> or <condition_2>:
❑ Not	if not <condition>:

❑ Pass: if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

```
if <condition>:
        pass
```

Loops

- Execute a block of code as long as a specified condition is reached.
- Loops are handy because they save time, reduce errors, and they make code more readable.
- While: <initialisation>

  while <condition>: <increment>
- For
  - for <var> in <list>:
  - for <var> in <string>:
  - for <var> in range(<end>): starting from 0, increments by 1, and ends at a specified number
  - for <var> in range(<start>, <end>):
  - for <var> in range(<start>, <end>, <increment>):
- Else in For Loop: specifies a block of code to be executed when the loop is finished
- Nested loops: loop inside a loop
- break statement: can stop the loop even if the while condition is true
- continue statement: can stop the current iteration, and continue with the next

Functions

- ❑ Creating a Function:

  def <fun_name>():

- ❑ Calling a Function:

  <fun_name>()

- ❑ Parameters/Arguments:

  def <fun_name>(<a1>, ... , <an>):

  Parameter is in function definition. Argument is sent to the function when it is called.

- ❑ Number of Arguments should be same in both definition and calling
- ❑ Default Parameter Value:

  def <fun_name>(<arg1> = <value1>):

- ❑ Arbitrary Arguments, *args: Don't know how many arguments that will be passed to function, add a * before the parameter name in the function definition (args[0] ⇒ first argument).
- ❑ Return Values

Functions (Cont…)

- Keyword Arguments: send arguments with the *key* = *value* syntax

```
def <fun_name>(<a1>, <a2>):
          <definition_body>
<fun_name>(<a2> = <value2>, <a1> = <value1>)                 #can pass arguments in any order
```

- Arbitrary Keyword Arguments, **kwargs: Don't know how many Keyword arguments will be passed, add two asterisk: ** before the parameter name in the function definition (args[<key_n>] = <value_n>).
- Passing a List as an Argument (can have loop to get values inside list)
- The pass Statement: function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.
- Keyword-Only Arguments: a function can have only keyword arguments (add *, **before** the arguments).

```
def <fun_name>(*, <a1>):
          <definition_body>
<fun_name>(<a1> = <value1>)                 # <fun_name>(<value1>)   will be an error
```

- **Recursion:** a defined function can call itself

Built-in Functions for Collection data types

- **Common:**
  - Length: len(<collection>)
  - Type: type(<collection>)
- **List:**
  - list() Constructor: <L1> = list((<a1>, <a2>, <a3>))
  - Access Items: L1[<index>]
  - Negative Indexing: means start from the end: -1 refers to the last item, -2 to the second last item
  - Check if Item Exists: if <a1> in <L1>:
  - Change Item Value: L1[<index>] = <value>
  - Change a Range of Item Values: L1[<start>:<stop>] = [<v1>, …… , <vN>]

    #The length of the list will change when the number of items inserted does not match the number of items replaced.
  - Insert Items: <L1>.insert(<index>, <value>)

    <L1>.append(<value>
  - Extend List: <L1>.extend(<L2>) #The extend() method does not have to append lists, you can add any iterable object (tuples, sets, dictionaries etc.).

Built-in Functions for Collection data types (Cont…)

- **List(Cont…):**
  - Remove Specific Item: <L1>.remove(<a1>)
  - Remove Specific Index: <L1>.pop(<index>) #If you do not specify the index, removes the last item
    del <L1>[<index>]
  - Clear the List: L1.clear() #clear the list
  - Loop Through a List: for <var> in <L1>:
  - Loop Through the Index Numbers: for <var> in range(len(<L1>)):
  - Looping Using List Comprehension: [print(x) for x in <L1>]
  - List Comprehension:List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

    Syntax: *newlist = [expression for item in iterable if condition == True]*

    Example: Based on list of fruits, you want a new list, containing only fruits with letter "a" in name.

    <new_list> = [x for x in fruits if "a" in x]

Built-in Functions for Collection data types (Cont…)

- **List(Cont…):**
  - Sort List Alphanumerically: <L1>.sort() #to sort descending, use reverse = True
  - Reverse Order: <L1>.reverse()
  - Copy a List: <L2> = <L1>.copy()
- **Tuple:**
  - Create Tuple With One Item: T1 = (<a1>,)#without comma it is just a string
  - tuple() Constructor: T1 = tuple((<a1>, ….. , <aN>))
  - Access Items & Negative Indexing: same as list
  - Update Tuples: convert tuple to list and update items

    ```
    L = list(T)
    L[<index>] = <new_val>
    T = tuple(L)
    ```
  - Tuple methods: count(), index()

Built-in Functions for Collection data types (Cont…)

- **Sets:**
  - set() Constructor: S1 = set((<a1>, ….. , <aN>))
  - Access Items: can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword

```
for x in S1:
    print(x)
```

  - Add Items: S1.add(<val>)
  - Add Sets: S1.update(S2)
  - Remove Item: S1.remove(<a1>) #If the item does not exist, remove() will raise an error.

    S1.discard(<a1>) #If the item does not exist, discard() will NOT raise an error.
  - Removes all the elements: S1.clear()
  - Copy set: S2 = S1.copy()
  - difference(): S3 = S1.difference(S2) #item in S1 that not in S2

MANAS RAHMAN

Built-in Functions for Collection data types (Cont…)

- **Dictionaries:**
  - dict() Constructor: D1 = dict(<k1> = <v1>, <k2> = <v2>, <k3> = <v3>)
  - Accessing Items: D1[<k1>] D1.get(<k1>)
  - Get Keys: D1.keys()
  - Get Values: D1.values()
  - Get Items: will return each item in a dictionary, as tuples in a list D1.items()
  - Check if Key Exists: if <k1> in D1:
  - Change Items: D1[<k1>] = <new_v1> D1.update({<k1>:<new_v1>})
  - Adding Items: is done by using a new index key and assigning a value to it
  - Removing Items: D1.pop(<k1>) D1.popitem() #last item in >= 3.7

    del D1[<k1>] del D1 #removes D1

    D1.clear()
  - Copy a Dictionary: D2 = D1.copy() D2 = dict(D1)
  - Nested Dictionaries: myfamily = {"child1" : {"name" : "Emil", "year" : 2004}, "child2" : {"name" : "Tobias","year" : 2007} or

    child1 = { "name" : "Emil", "year" : 2004 } child2 = { "name" : "Tobias", "year" : 2007 } myfamily = { "child1" : child1, "child2" : child2}

    print(myfamily["child2"]["name"]) #Access Items in Nested Dictionaries

MANAS RAHMAN

Arrays

- ❑ Python does not have built-in support for Arrays, but Python Lists can be used instead.

  Eg:

  ```
  <arr_name> = [<a1>, ……. , <aN>]
  <arr_name>[<index>]          #to access/modify elements (starting from 0)
  ```

- ❑ Length of an Array: `len(<arr_name>)`
- ❑ Looping Array Elements: `for <var> in <arr_name>:`
- ❑ Adding Array Elements: `<arr_name>.append(<element>)`
- ❑ Removing Array Elements: `<arr_name>.pop(<index>)` or `<arr_name>.remove(<element>)`
- ❑ Other Functions:

  ```
  <arr_name>.clear()   #Removes all the elements          <arr_name>.copy()   #Returns a copy
  <arr_name>.count(<element>)   #Number of elements with the specified value
  <arr1>.extend(<arr2>)   #Add the elements of a list (or any iterable), to the end of the current list
  <arr_name>.index(<element>)   #Returns the index of the first element with the specified value
  <arr_name>.insert(<index>, <element>)   #Adds an element at the specified position
  <arr_name>.reverse()   #reverse the order          <arr_name>.sort()   #reverse=True (sort in descending order)
  ```

Matrix Programs

Addition:

```
X = [[12,7,3], [4 ,5,6], [7 ,8,9]]
Y = [[5,8,1], [6,7,3], [4,5,9]]
result = [[0,0,0], [0,0,0], [0,0,0]]
for i in range(len(X)):                         # iterate through rows
    for j in range(len(X[0])):                  # iterate through columns
        result[i][j] = X[i][j] + Y[i][j]
for r in result:
    print(r)
```

Transpose:

```
X = [[12,7], [4 ,5], [3 ,8]]
result = [[0,0,0], [0,0,0]]
for i in range(len(X)):                         # iterate through rows
    for j in range(len(X[0])):                  # iterate through columns
        result[j][i] = X[i][j]
for r in result:
    print(r)
```

Multiplication:

```
X = [[12,7,3], [4 ,5,6], [7 ,8,9]]
Y = [[5,8,1,2], [6,7,3,0], [4,5,9,1]]
result = [[0,0,0,0], [0,0,0,0], [0,0,0,0]]
for i in range(len(X)):                         # iterate through rows of X
    for j in range(len(Y[0])):                  # iterate through columns of Y
        for k in range(len(Y)):                 # iterate through rows of Y
            result[i][j] += X[i][k] * Y[k][j]
for r in result:
    print(r)
```

Scope

- ❑ A variable is only available from inside the region it is created. This is called scope.
- ❑ Local Scope:
  - ❑ Inside Function: Variable created inside a function belongs to local scope of that function, and can only be used inside that function.
  - ❑ Function Inside Function: A variable is not available outside the function, but it is available for any function inside the function.
- ❑ Global Scope:
  - ❑ A variable created in the main body of the Python code is a global variable and belongs to the global scope.
  - ❑ Global variables are available from within any scope, global and local.
- ❑ Naming Variables:
  - ❑ If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope and one available in the local scope.
- ❑ Global Keyword:
  - ❑ If you need to create a global variable, but are stuck in the local scope, you can use the global keyword

    Eg: global <var_name>
  - ❑ Use the global keyword if you want to make a change to a global variable inside a function

    Eg: global <var_name> <var_name> = <new_value>

Programs

1. Program to find the sum of digits in any given number.
2. Program to count the occurrences of each word in a line of text.
3. Program to display future leap years from the current year to a final year entered by the user.
4. Program to use list comprehension to find the square of N numbers inside the below list. li = [25, 13, 12, 30, 40, 77, 88, 99]
5. Program to perform matrix addition of any given two matrices with any dimension.
6. Program to find the factorial of a number using loops and also using recursion.
7. Program to find prime numbers in any given range.
8. Program to find gcd of 2 numbers.
9. Program to find armstrong numbers in any given range.
10. Program to find area of circle, square, rectangle using functions by getting the required values as input from the user.
11. Program to construct a pyramid of stars with user specified height.
12. Generate all factors of a number
13. Find the sum of all items in a list

THANK YOU

MANAS RAHMAN