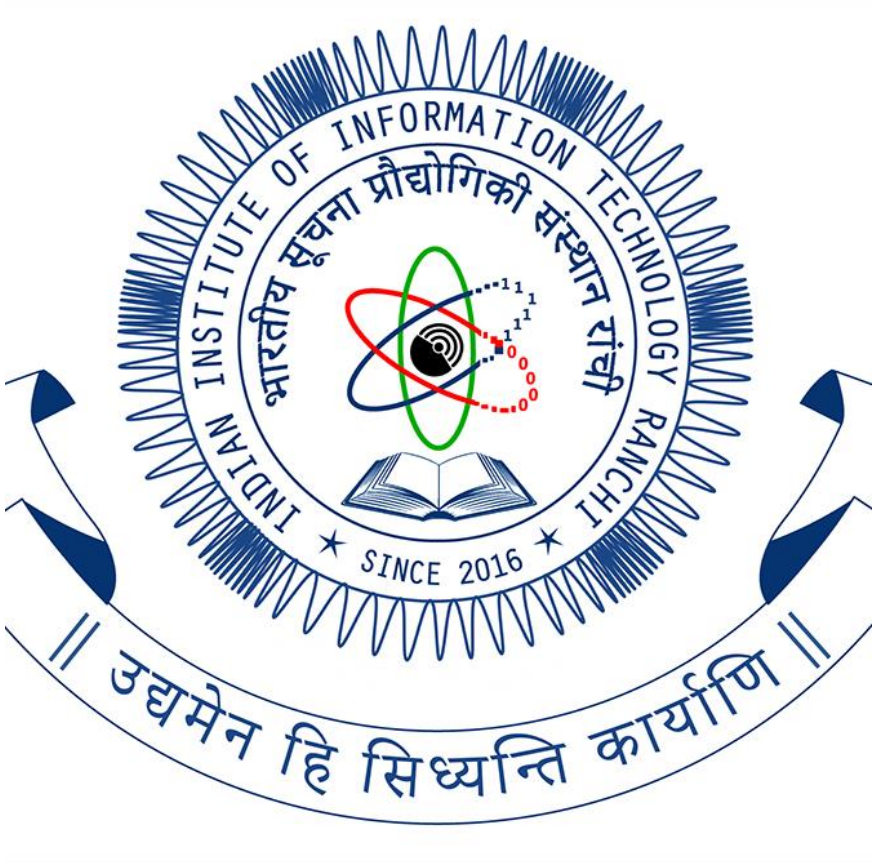
Python Programming

CS2001

Lecture-2: Python Basics



+

python™

Shivang Tripathi

Overview

- **Python Identifiers**
  - Naming conventions
- **Keywords**
- **Variables**
- **Data Types in Python**
  - Int
  - Float
  - complex
  - Bool
  - Str
- **String: String operations**
  - String formatting
- **Data Input from Console/Terminal by user**
- **Python operators**

Python Basics: Identifiers

- Identifiers are names used to identify variables, functions, classes, modules, and other objects in Python.
- Rules to remember
  - **Start with a Letter (Upper case or Lower case) or Underscore (_):**
    - Example: myVar, _privateVar
    - Invalid Example: 2variable (cannot start with a number)
  - **Followed by Letters, Numbers, or Underscores:**
    - Example: var123, my_var, name2
    - Invalid Example: var@name
      - (special characters !,@,#,$,% etc., aren't allowed)

Identifiers

- **Case-Sensitive:**
  - myVar and myvar are different identifiers.
- **Cannot Use Python Reserved Keywords:**
  - Example: def, class, if, etc., are reserved and cannot be used as identifiers.
  - Invalid Example: class = 10 (using reserved keyword as an identifier)

Naming conventions

- Best Practices for Naming Identifiers
  - **Use Descriptive Names:**
    - Example: total_sum, student_age (improves code readability)
  - **Avoid Single-Letter Names** (except for loop counters or simple purposes):
    - Example: Prefer count over c.
  - **Consistency in Naming Conventions:**
    - camelCase: Typically used for classes (e.g., myClass).
    - snake_case: Typically used for variables and functions (e.g., calculate_sum).

Keywords

- Keywords are reserved words in Python that have special meanings and are used to define the syntax and structure of the language.
- Keywords are case sensitive and are always in lowercase
- Keywords are reserved for exclusive use of interpreter, thus cannot be used as variable names, function names, or any other identifiers.
- To see all keywords in python use the following commands
    - **`import keyword`**
    - **`print(keyword.kwlist)`**

Keywords

Common Python Keywords:

- **Control Flow:**
  - if, else, elif
  - for, while
  - break, continue, pass
- **Data Handling:**
  - True, False, None
  - and, or, not
  - in, is
- **Function and Class Definitions:**
  - def, return
  - class, lambda
- **Exception Handling:**
  - try, except, finally
  - raise, assert
- **Variable Scoping:**
  - global, nonlocal
- **Miscellaneous:**
  - import, from, as
  - with, yield
  - del, await

Variable

- A variable in programming is a symbolic name or label that refers to a location in memory where data can be stored, modified, and retrieved.
- In Python, a variable is created when you assign a value to it
  - using the '=' sign.
  - E.g.: `x = 10` #x is a variable holding the integer value 10
- Python determines the type of the variable automatically based on what data is assigned to it

Data Types in Python

- Dynamic datatype assignment in Python
- Type of the variable is determined at run time
- High flexibility for variable initialization
- Datatype defines the memory requirement of the variable
- Datatype can be
  - Primary – int, float, complex, bool, str
  - Secondary or container type – list, dictionary, tuple, set
  - User defined -classes

Numeric data types:

- Numbers in python are represented by:
  - Integers (int) : x = 10
  - Floating points (float): x = 10.5, 105e-1
  - Complex (complex): x = 1 + 2j
  - Boolean (bool): True or False
- Numeric Literal: 2
- Numeric Variable: num = 2
- Immutable

All basic data types are built-in class defined in python
- Variables are objects of that class
- X = 10 creates an object of type class 'int'
- type(x)

Note: Python 2.x had two types for integers: int & long
- Long is obsolete in 3.x
- There is no longer limit to the value of integer

Numeric data types:

Complex Numbers

- Ordered pair of numbers: x + yj Note: x & y could be int or float
- Function complex() creates a complex number

```
>>> z = complex(2, 3)
2 + 3j
>>> z.real
>>> z.imag
>>> z.conjugate()  # conjugate() method used to get conjugate
>>> abs(2 + 3j)    # to get the absolute of the complex number
```

Numeric data types:

Boolean

- True and False data can be stored as bool objects.
- Arithmetic and logical operations are possible on bool objects
- All non-zero numbers are treated as True
- Zero is treated as False

```
>>>print(bool(5))
>>>print(bool(-6))
>>>print(bool(0))
>>>print(bool('abc'))
>>>print(bool(''))
>>>print(bool(' '))
>>>print(bool([]))
>>>print(bool([1,2]))
```

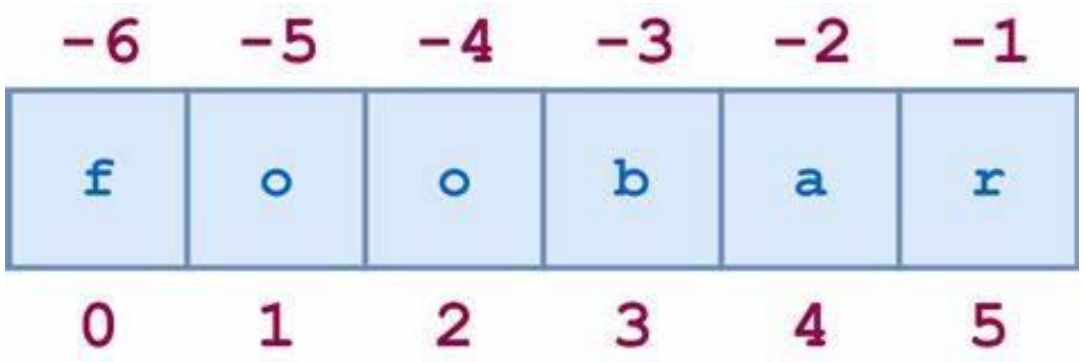
Type conversion

- The process of converting one data type to another:
  - **Implicit Conversion**: Automatically done by Python
    - Ex: sum = 2 + 3.2 -> interpreter will convert 2 to float and resultant sum will also be a float type
  - **Explicit Conversion**: Manually done by the user using specific functions
    - int() : Converts a value to an integer
    - float() : Converts a value to a float
    - str() : Converts a value to a string
    - list() : Converts a sequence (like a tuple) to a list
    - tuple() : Converts a sequence (like a list) to a tuple.
- Division of integer by another integer leads to float
- Python supports mixed arithmetic
  - If two operand are of diff. type (int & float) then narrower type is converted to wider type [n->w]:[int, float, complex]

Strings: str

- A string is a sequence of characters enclosed in quotes (single, double, or triple).
  - Example: a = 'Hello' or "Hello" or '''Hello''' Try: type(a)
- String is an ordered collection –
  - Elements are stored in the order in which they are inserted
  - Elements can be accessed using an index
- Strings are iterable -
- Strings are immutable -

Indexing



String indexing can be positive or negative

Positive indexing starts from the first letter of the string and begins with 0 index

Negative indexing starts from the last letter of the string and begins with -1 index

String operations

- Slicing: Create substrings – format: [start:stop:step]
  - Step is optional (default value is 1)
  - Slicing extracts characters from index start to end-1.
  - String1= "Shakespeare"
  - sub1=String1[0:5] => Shake
  - Sub2=String1[0:6:2] => ?
  - sub3 =String1[::1] – start at 0 end at the last letter with step 1 – replication of the string
  - Sub4=String1[::-1]
  - Sub5 = String[-2]

String operations

- Concatenation: joining two or more strings using '+'
  - Hello = 'Hello' + 'Python' + 'learners'
- String Repetition: repeating a string multiple times '*'
  - A = "Hello" * 3
- Type conversion
  - str() : fxn converts passed argument to type str

String functions & Methods

- len(): function returns the length of the string
- lower() and upper(): Converts the string to lowercase or uppercase
- strip(): Removes whitespace from the beginning and end of the string
- replace(): Replaces a substring with another substring
- split(): Splits the string into a list of substrings
- join(): Joins elements of a list into a single string
- dir(str) for more…

```
text = "  Hello, Python!  "
print(len(text))              # Result: 17
print(text.lower())           # Result: "  hello, python!  "
print(text.strip())           # Result: "Hello, Python!"
print(text.replace("Python", "World"))  # Result: "  Hello, World!  "
```

String formatting

- create and manipulate strings by inserting variables, expressions, or values.
- Old Style (%) String Formatting
  - Syntax: "string with %s placeholders" % (values)
    - Placeholders:
      - %s – String or any object with a string representation.
      - %d – Integers
      - %f – Floating-point numbers (%.2f - precision for float no.)
      - %x – Hexadecimal integers

```
name = "Alice"
age = 30
info = "Name: %s, Age: %d" % (name, age)
print(info)
Output:
Name: Alice, Age: 30
```

String formatting

- create and manipulate strings by inserting variables, expressions, or values.
- str.format() Method
    - Syntax: "string with {} placeholders".format(values)

```
name = "Alice"
age = 30
info = "Name: {}, Age: {}".format(name, age)
print(info)
==========================================
info = "Name: {0}, Age: {1}".format(name, age)
# or using keyword arguments
info = "Name: {name}, Age:{age}".format(name="Alice", age=30)
print(info)
==========================================
pi = 3.14159
formatted_pi = "Pi: {:.2f}".format(pi)
print(formatted_pi)
```

String formatting

- F-strings (Formatted String Literals) – Python 3.6+
- F-strings provide a concise and readable way to embed expressions inside string literals, using curly braces { }.
- Syntax: f"string with {expression} placeholders"

```
name = "Alice"
age = 30
info = f"Name: {name}, Age: {age}"
print(info)
====================================
pi = 3.14159
formatted_pi = f"Pi: {pi:.2f}"
print(formatted_pi)
```

…And few more ways to do the same thing…

Bonus:
Use Raw Strings if your string has many \
print(r'The path is C:\User\Documents')

Container type – list, dictionary, tuple, set

- Container in python can contain other objects (any valid python objects)
- **List:** contains items separated by commas & enclosed with [ ]
  - E.g. myList = [10, 10.5, 'python']
- **Tuple:** similar to list but with parentheses ( )
  - Unlike list, tuples are immutable
  - E.g. myTuple = (1, 2, 3, 4)
- **Dictionary:** contains unordered key-value pairs (keys are unique)
  - Enclosed by curly brackets { }
  - E.g. myDict = {'A' : 'Apple', 'B': 'Banana'}
- **Set:** contains unordered collection of immutable & unique objects
  - Can't have multiple occurrences of same element

Data input

- Python 3 version by default supports entry of string through keyboard

```
• Data=input("Enter the number\n")
12
• print(Data)
'12'
- type(Data)??
• Data=int(input("enter the number\n"))
12
• print(Data)
12
- Type(Data)??
```

- Try with Data = float(input(…))

Python Operators

Python operators are special symbols or keywords that are used to perform operations on variables and values.

Types of Operators

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

Arithmetic Operators

Operator	Description	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division (returns a float)	x / y
//	Floor Division (returns an integer)	x // y
%	Modulus (remainder of the division)	x % y
**	Exponentiation (power)	x ** y

Note: In python, sign of Remainder is same as the sign of denominator (unlike C)

Comparison (Relational) Operators

Comparison operators are used to compare two values.

- They return True or False based on the condition.

Operator	Description	Example
==	Equal to	x == y
!=	Not equal to	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 and y
or	Returns True if one of the statements is true	x or y
not	Reverses the result, returns False if the result is true	not x

Bitwise Operators

Bitwise operators are used to perform bit-level operations on integers.

These operators treat numbers as a sequence of bits.

Operator	Description	Example
&	AND	x & y
|	OR	x | y
^	XOR (Exclusive OR)	x ^ y
~	NOT (Inverts all bits)	~x
<<	Left shift (shift bits left)	x << 2
>>	Right shift (shift bits right)	x >> 2

Introduce bin, hex numbers

Assignment Operators

Assignment operators are used to assign values to variables.

Operator	Description	Example
=	Assigns right side value to the left side	x = 5
+=	Adds right side value to left and assigns	x += 5
-=	Subtracts right side value from left and assigns	x -= 5
*=	Multiplies left side by right side and assigns	x *= 5
/=	Divides left side by right side and assigns	x /= 5
//=	Floor divides left side by right side and assigns	x //= 5
%=	Takes modulus using two operands and assigns	x %= 5
**=	Performs exponentiation and assigns	x **= 5

Short Hand operators

Membership Operators

Membership operators are used to test whether a value or variable is found in a sequence (like a string, list, tuple, etc.).

Operator	Description	Example
in	Returns True if the value is in the sequence	x in y
not in	Returns True if the value is not in the sequence	x not in y

Identity Operators

Identity operators are used to compare the memory locations of two objects

Operator	Description	Example
is	Returns True if both variables point to the same object	x is y
is not	Returns True if both variables do not point to the same object	x is not y

Bonus: Conditional Expression

value_if_true **if** condition **else** value_if_false

```
x = 5; y = 10
result = "x is greater" if x > y else "y
is greater"
print(result)
```

Operator Precedence

determines the order in which operations are performed in expressions.

```
()
**
*,/,//,%
+,-
```

Same Precedence

Same Precedence

Associativity

When two operators have the same precedence level, the **associativity** of the operators determines the order of evaluation.

- Left-to-Right (Left Associativity)
- Right-to-Left (Right Associative)

Quiz = 10 + 2 * 3 ** 2 / 2 - 1
print(Quiz) ??

Operator	Description	Associativity
**	Exponentiation	Right-to-Left
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT	Right-to-Left
*, /, //, %	Multiplication, Division, Floor division, Modulus	Left-to-Right
+, -	Addition, Subtraction	Left-to-Right
<<, >>, &, ^, I	Bitwise operators	Left-to-Right
==, !=, >, <, >=, <=, is, is not, in, not in	Comparisons	Left-to-Right
not	Logical NOT	Right-to-Left
and	Logical AND	Left-to-Right
or	Logical OR	Left-to-Right