LECTURE - 1

7/10/21 - M

9/10/21 - N

Python was developed by Guido Van Rossum in 1991 at National research institute for Mathematics & Computer Science in the Netherlands.

It has many outstanding features :-

1. Easy language : This means that it is easy to read, write, learn and understand
   - It has a smooth learning curve
   - Due to its simple syntax it is easy to write and easy to understand.
   - Compared to other languages we'll see that reading someone's python code is easier rather than reading code of some other programming language.

2. Readable : it's like reading an english sentence.
   - it uses indentation instead of curly braces.

3. Interpreted language : It comes with IDLE (Interactive Development Environment). This is an interpreter & follows the REPL structure (Read Evaluate print Loop)

4. Dynamically - Typed Language : i.e., it is not statically type like some other languages.

   We don't need to declare data type while defining variable. Interpreter determines it at the runtime.

5. Open - Source : It is an open-source & community is always contributing it to improve it. Also, it is free and its source code is freely available to the public.

6. Large standard library : has many packages and modules with common & important functionality. So, if something is available in library we don't need to write it from scratch and focus on other important things.

7. Platform independent : It will run on different platforms like Windows, Mac, Linux. We don't need to write them separately on each other platform.

8. Extensible and Embeddable : → You can embed your Python code in other languages like C++.
↳ We can use code from other languages like C++ in your python code

Execution of Python program

2 steps

(1) Compilation

The program is converted into byte code internally. We can also view it.

↓ Is a fixed set of instructions that represents arithmetic, comparison, memory operation etc.

It can run on any operating system & hardware.

(2) Interpreter

Converts byte code into machine code. Python virtual machine (PVM) first understands OS & processor in the computer & then converts it into machine code. The machine code is executed by the processor & results are displayed.

Python Variables (In pythonic style variables are names & assignment is association) ②

- ∵ it is dynamically typed language hence we won't declare variable type.

directly print a;b , a,b
(a;b) This will run is value assigned to a & b.
(a,b) returns tuple is string

- A variable is created when we ~~declare~~ assign a value to it 1st time.
- can only contain no., letters, _ underscore.
- must begin with _ or letter.
- case sensitive.
- can not start with a number.

M - soa_sec_m2019@googlegroups.com
N - itercsen2023@googlegroups.com

examples :- (i) k_m , _km , _k2 , k_2_a , abc are all Valid.

(ii) km@, 2_km , # km , km* are not valid

11/10 - N

Arithmetic Operators

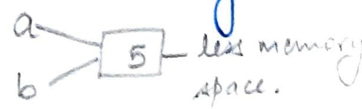
+ , − , * , /,%, // , (**) → exponential operator

+ → Addition
− → subtraction
* → multiplica-tion
/ → divides the no. and data type is ~~int~~ float
// → datatype is ~~float~~ True
(**) → raising to power of base no.

8/10/21 - M

- Also, discuss all these operations by taking input as string & numeric value.

more than one variable can refer to the same object



+ : string and string can be added , string & numeric can't be added.

assigning same value to multiple variables like a = b = 0

a = a + b is same as a += b
short-hand notation

//, /, −, % : only numeric values can be subtracted

* : string * numeric value = string. string. . . . . . string.
↳ numeric value times

- spaces are not acceptable while defining variable.
- while concatenating (string ,string), (string, numeric value) space is not included

LECTURE - 2

Execution of a python program

We save the file in script mode with (.py) extension in python(.pyw) can be used.

While compiling python program we cannot see .pyc file produced by the compiler & the machine code generated by (PVM).

→ We can compile it in console window using

```
python filename.py
```

python ↓ command for calling python compiler

- After this compiler directly displays the result, rather than converting it into byte code file i.e., pyc.

- In order to create .pyc file from source code, we use the command

```
python -m py-compile filename.py
```

→ Now in order to interpret we can again call the compiler using `python filename.cpython-38.pyc` command.

IDLE — Interpreter for python

(Interactive interpreter) Python Shell — Python Editor (Allows to work in script mode)

- type code at the prompt & press enter
- used as a calculator

→ '+, -, /, //, %, *' — left associative operators

→ '**' — right associative

→ talk about wrongly defined commands as python shows error.

eg: (i) 3/0 , (ii) 7 + 2(3+5) (syntax error)

→ (), **, -(negation), /, //, *, %(modulus), +, - equivalent precedence.

Strings (Sequence of characters)

eg: 1) How 'do' you "do" ? (check it in class by students)

2) "Hello" (Try using triple inverted commas)

- We enclose strings in ' ', " ", ''' ''', """ """ .
- ' ' includes " " and vice-versa.
- ''' ''' or """ """ are called documentation strings or doc. strings.
- There should not be blank line above or below the documentation string.
- \n marks new line

# Keywords are reserved and are already defined in python for specific uses, hence can't be used for other purposes.

⇒ We can't use keywords as variable names or for defining objects.

Relational Operators

- Compares two expressions and yields true or false.
- ==, <, >, <=, >=, != are relational operators. (equal to; not equal to)

eg: 23!=23 False

- String values can not be compared with numeric values, by default it give False.
- In order to access the ASCII code we use ord() funcⁿ.

eg: ord ('h') = 104
ord ('a') = 97
ord ('^') = 94

- If we want to decode the char. from encoded code we use chr() funcⁿ.

eg: chr(104) = h
chr (97) = a
chr (94) = ^

- Basically, relational operators test or defines some kind of relation b/w two entities.
- Two strings can be compared. eg: 'x'<'y' - True

Logical operators

- 'not', 'and' & 'or' are logical operators applied to logical operands (True & False called Boolean values) & yeild true & false.

unary operator as it requires only one entry

for True → False & False → True

'and' & 'or' are binary. (from slide see the table)

<u>Precedence</u> :- not ⟶ and ⟶ or

decreasing order

Precedence # Arithmetic operator > Relational operator > Logical operator

eg :- $\underbrace{10 < 5}$ and ((5/0)<10)

False , now it won't check the 2nd term as when any one part of and operator is false, output is always false. So, it will save time of the compiler.

whereas,

$\underbrace{(10 > 5)}$ and ((5/0)< 10)

True i.e., we need to check the 2nd term & ∵ 2nd term is not properly defined it will show an error.

Now, tell the output of ((5/0) < 10) and (10 > 5)

Logical eg (a) (i) 'p' or 1

(True) → 'p'

(ii) 'p' or 'k'

'p'

(iii) 1 or 'p'

1

because 0 is false [illegible]

(iv) 0 or 'p'

'p'

(v) 'p' or 0

'p'

(b) (i) 'p' and 1

1

(ii) 'p' and 'k'

'k'

(iii) 1 and 'p'

'p'

(iv) 0 and 'p'

0

(v) 'p' and 0

0

(c) (i) not 1 False

(ii) not 0 True

(iii) not p False

Bitwise Operators

• Using these operators integers are interpreted as strings of binary digits 0 and 1. also called bits.

Bitwise And (&) : x & y is 1 if the corresponding bit is 1 in each of x & y otherwise 0.

Q: Diff. b/w logical & bitwise

eg: 10 :- 00001010
8 :- 00001000

10 & 8 = 00000010 = 2

Bitwise OR (|) : if any one bit of x & y is 1 then 1 otherwise 0

eg: 10|8 = 00001110

Left shift (<<) : fill vacant spaces by 0 & result is $x * 2^y$
(x<<y)

eg: 5<<2 &5 = 00000101

= 00010100 = 20

Right shif (>>) : eg: 5>>2 result is $\frac{x}{2^y}$
(x>>y)

= 00000001 = 1

Bitwise NOT (~) : 1 is made 0 & 0 is made 1 and output value is = -(n+1)
or complement

eg: 11 = 00001011 = 11110100 = -(11+1) = -12

Bitwise exclusive OR(^) : if exactly one bit of x or y is 1 then 1 otherwise 0.
XOR

eg: 10^6 , 10 = 00001010 , 6 = 00000110

= 00001100 = 12

Conversion :- (i) 13_{10}

$$\begin{array}{r|rr} 2 & 13 & 1 \\ 2 & 6 & 0 \\ 2 & 3 & 1 \\ & 1 & 1 \end{array} = (\overset{2^3}{1}\overset{2^2}{1}\overset{2^1}{0}\overset{2^0}{1})_2$$

switch on   switch off

now to get back to decimal form

$= 2^0 + 2^2 + 2^3 = 1+4+8$

$= 13$

Byte code and Machine code (Doubt of some student)

Byte code:
- Machine independent code. (contained in .pyc)
- PVM converts it into machine code because of it, we say that python is platform independent.

Machine code:
- Platform dependent i.e., they run only where they are compiled.
  eg: Code of Windows 2007 doesn't run on Mac
- 0's & 1's are used for source code conversion to machine code.

⇒ Python has two translators for
Feature:
① platform independent feature.
② programmer has to write code only once.

Drawback of this feature
- Code will run only when we have PVM on targeted OS.
- Slow speed.

#. Discuss the precedence of operators. (from slides)
- Strings are compared left to right character by character.
- If a string is a prefix of another string then larger string will be the longer one.
- In And logical operator & in or logical operator if 1st condition False and True respectively then it doesn't move to 2nd cond. This is called short-circuit evaluation.

eg: not (9 == 8) and (7+1 != 8) or (6 < 4.5)

Bitwise representation of negative numbers

−15 and 22

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	0	0	0	1	1	1	1	→ 15
128	64	32	16	8	4	2	1	

$\underbrace{8\ 4\ 2\ 1}_{=15}$

Now, in order to find ⊖ve number representation we need to find the two's complement in 8-bit representation. To find two's complement, we find one's complement and we know that one's complement is found by inverting every bit such as :-

1	1	1	1	0	0	0	0

+ 1

1	1	1	1	0	0	0	1	— Two's complement

= −15

Rules to add 1 :-

1+1 = 0 (carry 1)

1 + 0 = 1

0 + 1 = 1

0 + 0 = 0

Now, bit value representation of 22 is :-

0	0	0	1	0	1	1	0

Now, we'll operate 'and' bit by bit

0	0	0	1	0	0	0	0

2^4 = $\boxed{16}$

Now, operating using or operator

1	1	1	1	0	1	1	1

now, we can see that the sign bit is 1
This means that it is the bitwise representation of negative nos.

So. after taking the two's complement we'll get

0	0	0	0	1	0	0	0

+ 1

0	0	0	0	1	0	0	1

$= \boxed{-9}$

How to get the output in same line using multiple print comma

We can use print('   ', end="   ") (end='Joker')

→ This means end with space.
We can end our line accordingly
We can end with some string joker also

or print('   ') print('   ')

Using any of these two statements we can get the output in same line.

• end is by default newline character.