Expressions

Expressions

- An expression in python is an valid combination of operators, literals and variables.

The expressions in python can be of any type :

- ✓ Arithmetic Expressions,
- ✓ Relational Expressions,
- ✓ Logical Expression,
- ✓ Compound Expressions Etc.

Expressions

- ***Arithmetic expression*** *involve numbers (integers, floating-point number, complex numbers*

  *e.g., 2+5 ** 3, –8 * 6/5*

- ***Relational expression****, Expressions having literals and/or variables of any valid type*

  e.g., x>y, y<=z, z!=x, z==q, x<y>z, x==y!=z

Expressions

- ***Logical expression**, expression having literals and/or variables of any valid type.*

  e.g., a or b, b and c, a and not b, not c or not b

- ***String expressions**, Python also provides two string operator + and *, when combined with string operands and integers.*

  e.g., "and" = "then"

  *"and" * 2*

Operator	Description	Operator	Description
( )	Parentheses (grouping)	Not x	Boolean NOT
**	Exponentiation	And	Boolean AND
~x	Bitwise nor	Or	Boolean OR
+x, -x	Positive, negative (unary +, -)	<	Lesser than
*	Multiplication	<=	Lesser than equal to
/	Division	>	Greater than
//	Floor division	>=	Greater than equal to
%	Remainder	<>	Lesser than Greater than
+, -	Addition, Subtraction	!=	Not equal to
&	Bitwise AND	==	Equal to
^	Bitwise XOR	is	Is
|	Bitwise OR	is not	Is not

020 EXPRESSIONS

```
#MODULO OR REMAINDER

a,b=3,7

c=b%a

print(c)
```

Evaluating Relational Expressions

- All comparison operations in python have the same priority
- which is lower than that of any arithmetic operations.
- All relational expressions comparison yield Boolean values only
- i.e., true or false

021 RELATIONAL EXPRESSIONS

```
a=10

n=20

b=30

print(a<=n<=b)
```

Python Logical Operators

- Python logical operators are used to combine conditional statements, allowing you to perform operations based on multiple conditions.
- These Python operators, alongside arithmetic operators, are special symbols used to carry out computations on values and variables.

TRUTH TABLE

X	Y	X and Y	X or Y	not(X)	not(Y)
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

Evaluating Logical Expressions

- While evaluating logical expansions python follows these rules

i) the precedence of logical operators is lower than the Arithmetic operators so constituent Arithmetic sub expression if any is evaluated first and then logical operators are applied if any is evaluated first and then logical operators are applied

- So the over all result will be 5.

022 LOGICAL EXPRESSIONS

```
print(5<10)and(10<5)or(3<18)and not(8<18)
```

Type Casting
Type Conversion

Type Casting (Explicit Type Conversion)

- Type Casting is the process of converting data of one type to another.

There are two types of type conversion in Python.

- **Implicit Conversion** - automatic type conversion
- **Explicit Conversion** - manual type conversion

Python Implicit Type Conversion

- In certain situations, Python automatically converts one data type to another.
- This is known as implicit type conversion.
- Python promotes the conversion of the lower data type (integer) to the higher data type (float) to avoid data loss.

023 IMPLICIT CONVERSION

```
a=123
b=1.23
c=a+b
print("Value of ", c)
print("Data Type of ", type(c))
```

Explicit Type Conversion

- In Explicit Type Conversion, users convert the data type of an object to required data type.
- We use the built-in functions like int(), float(), str(), etc to perform explicit type conversion.
- This type of conversion is also called typecasting because the user casts (changes) the data type of the objects.

024 EXPLICIT CONVERSION

```
a='12'
b=23
print("Data Type of a Before Type Casting : ", type(a))
a=int(a)
print("Data Type of a After Type Casting : ", type(a))
c=a+b
print("Sum : ", c)
print("Data Type of c : ", type(c))
```

Math Library Functions

- Pythons standard library provides a module namely math for math related functions that work with all number types expect for complex numbers
- In other two work with functions of math module you need to first import it to your program by given statement is follows as the loop line of your python script
- Import math
- Then you can use math libraries functions as math <function name>

Mathematical Functions in Math module

S. No.	Function	Prototype (General Form)	Description	Example
1.	ceil	math.ceil(num)	The **ceil( )** function returns the smallest integer not less than *num*.	math.ceil(1.03) gives 2.0 math.ceil(–1.03) gives –1.0.
2.	sqrt	math.sqrt (num)	The **sqrt( )** function returns the square root of *num*. If num < 0, domain error occurs.	math.sqrt(81.0) gives 9.0.
3.	exp	math.exp(arg)	The **exp( )** function returns the natural logarithm **e** raised to the *arg* power.	math.exp(2.0) gives the value of e^2.
4.	fabs	math.fabs (num)	The **fabs( )** function returns the absolute value of *num*.	math.fabs(1.0) gives 1.0 math.fabs(–1.0) gives 1.0.
5.	floor	math.floor (num)	The **floor( )** function returns the largest integer not greater than *num*.	math.floor(1.03) gives 1.0 math.floor(–1.03) gives –2.0.
6.	log	math.log (num, [base])	The **log( )** function returns the natural logarithm for *num*. A domain error occurs if *num* is negative and a range error occurs if the argument *num* is zero.	math.log(1.0) gives the natural logarithm for 1.0. math.log(1024, 2) will give logarithm of 1024 to the base 2.

Mathematical Functions in Math module

7.	log10	math.log10 (num)	The **log10( )** function returns the base 10 logarithm for *num*. A domain error occurs if *num* is negative and a range error occurs if the argument is zero.	math.log10(1.0) gives base 10 logarithm for 1.0.
8.	pow	math.pow (base, exp)	The **pow( )** function returns *base* raised to *exp* power *i.e,. base exp*. A domain error occurs if *base* =0 and *exp* <=0 ; also if *base* <0 and *exp* is not integer.	math.pow(3.0, 0) gives value of 3^0. math.pow(4.0, 2.0) gives value of 4^2.
9.	sin	math.sin(arg)	The **sin( )** function returns the sine of *arg*. The value of *arg* must be in radians.	math.sin(val) (*val* is a number).
10.	cos	math.cos(arg)	The **cos( )** function returns the cosine of *arg*. The value of *arg* must be in radians.	math.cos(val) (*val* is a number).
11.	tan	math.tan(arg)	The **tan( )** function returns the tangent of *arg*. The value of *arg* must be in radians.	math.tan(val) (val is a number)
12.	degrees	math.degrees(*x*)	The **degrees( )** converts angle *x* from radians to degrees.	math.degrees(3.14) would give 179.91
13.	radians	math.radians(*x*)	The **radians( )** converts angle *x* from degrees to radians.	math.radians(179.91) would give 3.14

Mathematical Functions in Math module

14.	fmod	math.fmod(x, y)	This function returns the modulus resulting from the division of *x* with *y*. It is little different from *x* % *y* in the sense that it offers more precision, and returns the result with the sign of *x* (unlike *x* % *y*, which returns with sign of *y*). *fmod* is preferred for *floats*, because of higher precision while *x* % *y* is preferred for *integers*.	math.fmod(7.4, 3.3) gives value as 0.8000000000000007 math.fmod(7.4, 3) gives value as 1.4000000000000004 math.fmod(–7.4, 3) gives value as –1.4000000000000004 math.fmod(–8.3, 6) gives value as –2.3000000000000007
15.	factorial	math.factorial(arg)	This function returns factorial of *arg* as an integer. It will raise *ValueError* if *arg* is not integral or is negative.	math.factorial(4) gives value as 24 math.factorial(6) gives value as 720
16.	gcd	math.gcd(x, y)	This function returns the *greatest common divisor* of the integers *x* and *y*. If either *x* or *y* is nonzero, then the value of **gcd**(*x*, *y*) is the largest positive integer that divides both *x* and *y*. **gcd**(0, 0) returns 0.	math.gcd(8, 6) gives value as 2 math.gcd(28, 20) gives value as 4

025 IMPORT MATHS MODULE

```
print(math.factorial(4))
print(math.factorial(6))


print(math.gcd( *integers:5,7))
print(math.gcd( *integers:2,6))
```

PYTHON TEST – 1.8
EXPERSSIONS