**Python : Mathematical Function ( part 1)**

Python includes following functions that perform mathematical calculations.

|  |  |
| --- | --- |
| Sr.No. | Function & Returns ( description ) |
| 1 | abs(x) :  The absolute value of x: the (positive) distance between x and zero. Description The method **abs()** returns absolute value of **x** - the (positive) distance between x and zero. Syntax Following is the syntax for **abs()** method −  abs( x ) Parameters  * **x** − This is a numeric expression.  Return Value This method returns absolute value of x. Example The following example shows the usage of abs() method.  #!/usr/bin/python  print "abs(-45) : ", abs(-45)  print "abs(100.12) : ", abs(100.12)  print "abs(119L) : ", abs(119L)  When we run above program, it produces following result −  abs(-45) : 45  abs(100.12) : 100.12  abs(119L) : 119 |
| 2 | ceil(x) :  The ceiling of x: the smallest integer not less than x Description The method **ceil()** returns ceiling value of **x** - the smallest integer not less than x. Syntax Following is the syntax for **ceil()** method −  import math  math.ceil( x )  **Note** − This function is not accessible directly, so we need to import math module and then we need to call this function using math static object. Parameters  * **x** − This is a numeric expression.  Return Value This method returns smallest integer not less than x. Example The following example shows the usage of ceil() method.  #!/usr/bin/python  import math # This will import math module  print "math.ceil(-45.17) : ", math.ceil(-45.17)  print "math.ceil(100.12) : ", math.ceil(100.12)  print "math.ceil(100.72) : ", math.ceil(100.72)  print "math.ceil(119L) : ", math.ceil(119L)  print "math.ceil(math.pi) : ", math.ceil(math.pi)  When we run above program, it produces following result −  math.ceil(-45.17) : -45.0  math.ceil(100.12) : 101.0  math.ceil(100.72) : 101.0  math.ceil(119L) : 119.0  math.ceil(math.pi) : 4.0 |
| 3 | Cmp (x,y)  -1 if x < y, 0 if x == y, or 1 if x > y Description The method **cmp()** returns the sign of the difference of two numbers : -1 if x < y, 0 if x == y, or 1 if x > y. Syntax Following is the syntax for **cmp()** method −  cmp( x, y ) Parameters  * **x** − This is a numeric expression. * **y** − This is also a numeric expression.  Return Value This method returns -1 if x < y, returns 0 if x == y and 1 if x > y Example The following example shows the usage of cmp() method.  #!/usr/bin/python  print "cmp(80, 100) : ", cmp(80, 100)  print "cmp(180, 100) : ", cmp(180, 100)  print "cmp(-80, 100) : ", cmp(-80, 100)  print "cmp(80, -100) : ", cmp(80, -100)  When we run above program, it produces following result −  cmp(80, 100) : -1  cmp(180, 100) : 1  cmp(-80, 100) : -1  cmp(80, -100) : 1 |
| 4 | exp(x) :  The exponential of x: ex Description The method **exp()** returns returns exponential of x: ex. Syntax Following is the syntax for **exp()** method −  import math  math.exp( x )  **Note** − This function is not accessible directly, so we need to import math module and then we need to call this function using math static object. Parameters  * **x** − This is a numeric expression.  Return Value This method returns exponential of x: ex. Example The following example shows the usage of exp() method.  #!/usr/bin/python  import math # This will import math module  print "math.exp(-45.17) : ", math.exp(-45.17)  print "math.exp(100.12) : ", math.exp(100.12)  print "math.exp(100.72) : ", math.exp(100.72)  print "math.exp(119L) : ", math.exp(119L)  print "math.exp(math.pi) : ", math.exp(math.pi)  When we run above program, it produces following result −  math.exp(-45.17) : 2.41500621326e-20  math.exp(100.12) : 3.03084361407e+43  math.exp(100.72) : 5.52255713025e+43  math.exp(119L) : 4.7978133273e+51  math.exp(math.pi) : 23.1406926328 |
| 5 | fabs(x) :  The absolute value of x. Description The method **fabs()** returns the absolute value of x. Syntax Following is the syntax for **fabs()** method −  import math  math.fabs( x )  **Note** − This function is not accessible directly, so we need to import math module and then we need to call this function using math static object. Parameters  * **x** − This is a numeric value.  Return Value This method returns absolute value of x. Example The following example shows the usage of fabs() method.  #!/usr/bin/python  import math # This will import **math** module  print "math.fabs(-45.17) : ", math.fabs(-45.17)  print "math.fabs(100.12) : ", math.fabs(100.12)  print "math.fabs(100.72) : ", math.fabs(100.72)  print "math.fabs(119L) : ", math.fabs(119L)  print "math.fabs(math.pi) : ", math.fabs(math.pi)  When we run above program, it produces following result −  math.fabs(-45.17) : 45.17  math.fabs(100.12) : 100.12  math.fabs(100.72) : 100.72  math.fabs(119L) : 119.0  math.fabs(math.pi) : 3.14159265359 |
| 6 | Floor (x) :  The floor of x: the largest integer not greater than x Description The method **floor()** returns floor of **x** - the largest integer not greater than x. Syntax Following is the syntax for **floor()** method −  import math  math.floor( x )  **Note** − This function is not accessible directly, so we need to import math module and then we need to call this function using math static object. Parameters  * **x** − This is a numeric expression.  Return Value This method returns largest integer not greater than x. Example The following example shows the usage of floor() method.  #!/usr/bin/python  import math # This will import **math** module  print "math.floor(-45.17) : ", math.floor(-45.17)  print "math.floor(100.12) : ", math.floor(100.12)  print "math.floor(100.72) : ", math.floor(100.72)  print "math.floor(119L) : ", math.floor(119L)  print "math.floor(math.pi) : ", math.floor(math.pi)  When we run above program, it produces following result −  math.floor(-45.17) : -46.0  math.floor(100.12) : 100.0  math.floor(100.72) : 100.0  math.floor(119L) : 119.0  math.floor(math.pi) : 3.0 |
| 7 | Log (x) :  The natural logarithm of x, for x> 0 Description The method **log()** returns natural logarithm of x, for x > 0. Syntax Following is the syntax for **log()** method −  import math  math.log( x )  **Note** − This function is not accessible directly, so we need to import math module and then we need to call this function using math static object. Parameters  * **x** − This is a numeric expression.  Return Value This method returns natural logarithm of x, for x > 0. Example The following example shows the usage of log() method.  #!/usr/bin/python  import math # This will import **math** module  print "math.log(100.12) : ", math.log(100.12)  print "math.log(100.72) : ", math.log(100.72)  print "math.log(119L) : ", math.log(119L)  print "math.log(math.pi) : ", math.log(math.pi)  When we run above program, it produces following result −  math.log(100.12) : 4.60636946656  math.log(100.72) : 4.61234438974  math.log(119L) : 4.77912349311  math.log(math.pi) : 1.14472988585 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |