Operations on numbers

Operation	Description
x + y	Addition
x – y	Subtraction
x * y	Multiplication
x / y	Division
x // y	Truncating division
x ** y	Power (x^y)
x % y	Modulo (x mod y)
–x	Unary minus
+x	Unary plus

Shifting and bitwise logical operations on integers

Operation	Description
x << y	Left shift
x >> y	Right shift
x & y	Bitwise and
x | y	Bitwise or
x ^ y	Bitwise xor (exclusive or)
~x	Bitwise negation

Built-in functions applicable to all numeric types

abs(x)	Absolute value
divmod(x,y)	Returns (x // y, x % y)
pow(x,y [,modulo])	Returns (x ** y) % modulo
round(x,[n])	Rounds to the nearest multiple of 10-*n* (floating-point numbers only)

Comparison operators

Operation Description

x < y	Less than
x > y	Greater than
x == y	Equal to
x != y	Not equal to
x >= y	Greater than or equal to
x <= y	Less than or equal to

Augmented Assignment

Operation	Description
x +=y	x= x + y
x -=y	x= x - y
x *=y	x= x * y
x /=y	x= x / y
x //=y	x = x // y
x **=y	x= x ** y
x %=y	x= x % y
x &=y	x= x & y
x |=y	x= x | y
x ^=y	x= x ^ y
x >>=y	x= x >> y
x <<=y	x=x << y

Conversion functions

Function	Description
int(x [,base])	Converts x to an integer. base specifies the base if x is a string.
float(x)	Converts x to a floating-point number.
complex(real [,imag])	Creates a complex number.
str(x)	Converts object x to a string representation.
repr(x)	* Converts object x to an expression string.
format(x [,format_spec])	Converts object x to a formatted string.
eval(str)	* Evaluates a string and returns an object.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.

set(s)	* Converts s to a set.
dict(d)	Creates a dictionary. d must be a sequence of (key,value) tuples.
frozenset(s)	* Converts s to a frozen set.
chr(x)	Converts an integer to a character.
ord(x)	Converts a single character to its integer value.
hex(x)	Converts an integer to a hexadecimal string.
bin(x)	Converts an integer to a binary string.
oct(x)	Converts an integer to an octal string.

Boolean expressions

Operator	Description
x or y	If x is false, return y; otherwise, return x.
x and y	If x is false, return x; otherwise, return y.
not x	If x is false, return True; otherwise, return False.

Order of Evaluation

All operators except the power (**) operator are evaluated from left to right and are listed in the table from highest to lowest precedence. That is, operators listed first in the table are evaluated before operators listed later. (Note that operators included together within subsections, such as x * y, x / y, x / y, and x % y, have equal precedence.)

Order of Evaluation (Highest to Lowest)

Operator	Name
(...), [...], {...}	Tuple, list, and dictionary creation
s[i], s[i:j]	Indexing and slicing
s.attr	Attributes
f(...)	Function calls
+x, -x, ~x	Unary operators
x ** y	Power (right associative)
x * y, x / y, x // y, x % y	Multiplication, division, floor division, modulo
x + y, x – y	Addition, subtraction
x << y, x >> y	Bit-shifting
x & y	Bitwise and
x ^ y	Bitwise exclusive or
x | y	Bitwise or
x < y, x <= y, x > y, x >= y, x == y, x != y, x is y, x is not y, x in s, x not in s	Comparison, identity, and sequence membership tests
not x	Logical negation
x and y	Logical and
x or y	Logical or
lambda args: expr	* Anonymous function

Sequences

Operations and Methods Applicable to All Sequences

Item	Description
s[i]	Returns element i of a sequence
s[i:j]	Returns a slice
s[i:j:stride]	Returns an extended slice
len(s)	Number of elements in s
min(s)	Minimum value in s
max(s)	Maximum value in s
sum(s [,initial])	Sum of items in s, only for numeric types
all(s)	Checks whether all items in s are True.
any(s)	Checks whether any item in s is True.
s + r	Concatenates the two sequences
s * n, n * s	Makes n copies of s, where n is an integer
v1,v2…, vn = s	Variable unpacking (n==len(s) must hold)
x in s, x not in s	Membership
for x in s:	Iteration

Operations Applicable to Mutable Sequences

Item	Description
s[i] = v	Item assignment
s[i:j] = t	Slice assignment
s[i:j:stride] = t	Extended slice assignment
del s[i]	Item deletion
del s[i:j]	Slice deletion
del s[i:j:stride]	Extended slice deletion

List Methods

Method	Description
list(s)	Converts s to a list.
s.append(x)	Appends a new element, x, to the end of s.
s.extend(t)	Appends a new list, t, to the end of s.
s.count(x)	Counts occurrences of x in s.
s.index(x [,start [,stop]])	Returns the smallest i where s[i]==x. start and stop optionally specify the starting and ending index for the search.
s.insert(i,x)	Inserts x at index i.
s.pop([i])	Returns the element i and removes it from the list. If i is omitted, the last element is returned.
s.remove(x)	Searches for x and removes it from s.
s.reverse()	Reverses items of s in place.
s.sort([key [, reverse]])	Sorts items of s in place. key is a key function. reverse is a flag that sorts the list in reverse order. key and reverse should always be specified as keyword arguments.

String

String Methods

Method	**Description**
s.capitalize()	Capitalizes the first character.
s.center(width [, pad])	Centers the string in a field of length width. pad is a padding character.
s.count(sub [,start [,end]])	Counts occurrences of the specified substring sub.
s.decode([encoding [,errors]])	* Decodes a string and returns a Unicode string (byte strings only).
s.encode([encoding [,errors]])	* Returns an encoded version of the string (unicode strings only).
s.endswith(suffix [,start [,end]])	Checks the end of the string for a suffix.
s.expandtabs([tabsize])	Replaces tabs with spaces.
s.find(sub [, start [,end]])	Finds the first occurrence of the specified substring sub or returns -1.
s.format(*args, **kwargs)	* Formats s.
s.index(sub [, start [,end]])	Finds the first occurrence of the specified substring sub or raises an error.
s.isalnum()	Checks whether all characters are alphanumeric.
s.isalpha()	Checks whether all characters are alphabetic.
s.isdigit()	Checks whether all characters are digits.
s.islower()	Checks whether all characters are lowercase.
s.isspace()	Checks whether all characters are whitespace.
s.istitle()	Checks whether the string is a titlecased string (first letter of each word capitalized).
s.isupper()	Checks whether all characters are uppercase.
s.join(t)	Joins the strings in sequence t with s as a separator.
s.ljust(width [, fill])	Left-aligns s in a string of size width.
s.lower()	Converts to lowercase.
s.lstrip([chrs])	Removes leading whitespace or characters supplied in chrs.
s.partition(sep)	Partitions a string based on a separator string sep. Returns a tuple (head,sep,tail) or (s, "","") if sep isn’t found.
s.replace(old, new [,maxreplace])	Replaces a substring.
s.rfind(sub [,start [,end]])	Finds the last occurrence of a substring.
s.rindex(sub [,start [,end]])	Finds the last occurrence or raises an error.
s.rjust(width [, fill])	Right-aligns s in a string of length width.
s.rpartition(sep)	Partitions s based on a separator sep, but searches from the end of the string.
s.rsplit([sep [,maxsplit]])	Splits a string from the end of the string using sep as a delimiter. maxsplit is the maximum number of splits to perform. If maxsplit is omitted, the result is identical to the split() method.
s.rstrip([chrs])	Removes trailing whitespace or characters supplied in chrs.
s.split([sep [,maxsplit]])	Splits a string using sep as a delimiter. maxsplit is the maximum number of splits to perform.
s.splitlines([keepends])	Splits a string into a list of lines. If keepends is True, trailing newlines are preserved.
s.startswith(prefix [,start	Checks whether a string starts with prefix.

[,end]])	
s.strip([chrs])	Removes leading and trailing whitespace or characters supplied in chrs.
s.swapcase()	Converts uppercase to lowercase, and vice versa.
s.title()	Returns a title-cased version of the string.
s.translate(table [,deletechars])	* Translates a string using a character translation table table, removing characters in deletechars.
s.upper()	Converts a string to uppercase.
s.zfill(width)	Pads a string with zeros on the left up to the specified width.

String Formatting Conversions

Character	**Output Format**
d,i	Decimal integer or long integer.
u	Unsigned integer or long integer.
o	Octal integer or long integer.
x	Hexadecimal integer or long integer.
X	Hexadecimal integer (uppercase letters).
f	Floating point as [-]m.dddddd.
e	Floating point as [-]m.dddddde±xx.
E	Floating point as [-]m.ddddddE±xx.
g,G	Use %e or %E for exponents less than –4 or greater than the precision; otherwise, use %f.
s	String or any object. The formatting code uses str() to generate strings.
r	Produces the same string as produced by repr().
c	Single character.
%	Literal %.

Dictionaries

Methods and Operations for Dictionaries

Item	**Description**
len(m)	Returns the number of items in m.
m[k]	Returns the item of m with key k.
m[k]=x	Sets m[k] to x.
del m[k]	Removes m[k] from m.
k in m	Returns True if k is a key in m.
m.clear()	Removes all items from m.
m.copy()	Makes a copy of m.
m.fromkeys(s [,value])	Create a new dictionary with keys from sequence s and values all set to value.
m.get(k [,v])	Returns m[k] if found; otherwise, returns v.
m.items()	Returns a sequence of (key,value) pairs.
m.keys()	Returns a sequence of key values.
m.pop(k [,default])	Returns m[k] if found and removes it from m; otherwise, returns default if supplied or raises KeyError if not.
m.popitem()	Removes a random (key,value) pair from m and returns it as a tuple.
m.setdefault(k [, v])	Returns m[k] if found; otherwise, returns v and sets m[k] = v.
m.update(b)	Adds all objects from b to m.
m.values()	Returns a sequence of all values in m.

* Sets

Methods and Operations for Set Types

Item	**Description**
len(s)	Returns the number of items in s.
s.copy()	Makes a copy of s.
s.difference(t)	Set difference. Returns all the items in s, but not in t.
s.intersection(t)	Intersection. Returns all the items that are both in s and in t.
s.isdisjoint(t)	Returns True if s and t have no items in common.
s.issubset(t)	Returns True if s is a subset of t.
s.issuperset(t)	Returns True if s is a superset of t.
s.symmetric_difference(t)	Symmetric difference. Returns all the items that are in s or t, but not in both sets.
s.union(t)	Union. Returns all items in s or t.

Methods for Mutable Set Types

Item	**Description**
s.add(item)	Adds item to s. Has no effect if item is already in s.
s.clear()	Removes all items from s.
s.difference_update(t)	Removes all the items from s that are also in t.
s.discard(item)	Removes item from s. If item is not a member of s, nothing happens.
s.intersection_update(t)	Computes the intersection of s and t and leaves the result in s.
s.pop()	Returns an arbitrary set element and removes it from s.
s.remove(item)	Removes item from s. If item is not a member, KeyError is raised.
s.symmetric_difference_update(t)	Computes the symmetric difference of s and t and leaves the result in s.
s.update(t)	Adds all the items in t to s. t may be another set, a sequence, or any object that supports iteration.

Operations on sets

Operation	**Description**
s | t	Union of s and t
s & t	Intersection of s and t
s – t	Set difference
s ^ t	Symmetric difference
len(s)	Number of items in the set
max(s)	Maximum value
min(s)	Minimum value

File

File Methods

Method	Description
f.read([n])	* Reads at most n bytes.
f.readline([n])	Reads a single line of input up to n characters. If n is omitted, this method reads the entire line.
f.readlines([size])	Reads all the lines and returns a list. size optionally specifies the approximate number of characters to read on the file before stopping.
f.write(s)	Writes string s.
f.writelines(lines)	Writes all strings in sequence lines.
f.close()	Closes the file.
f.tell()	* Returns the current file pointer.
f.seek(offset [, whence])	* Seeks to a new file position.
f.isatty()	* Returns True if f is an interactive terminal.
f.flush()	* Flushes the output buffers.
f.truncate([size])	* Truncates the file to at most size bytes.
f.fileno()	* Returns an integer file descriptor.
f. __next__()	* Returns the next line or raises StopIteration.

File Object Attributes

Attribute	Description
f.closed	Boolean value indicates the file state: False if the file is open, True if closed.
f.mode	The I/O mode for the file.
f.name	Name of the file if created using open(). Otherwise, it will be a string indicating the source of the file.
f.softspace	* Boolean value indicating whether a space character needs to be printed before another value when using the print statement.
f.newlines	* When a file is opened in universal newline mode, this attribute contains the newline representation actually found in the file. The value is None if no newlines have been encountered, a string containing '\n', '\r', or '\r\n', or a tuple containing all the different newline encodings seen.
f.encoding	* A string that indicates file encoding, if any (for example, 'latin-1' or 'utf-8'). The value is None if no encoding is being used.

Built-in functions and types

abs(x)	Returns the absolute value of x.
all(s)	Returns True if all of the values in the iterable s evaluate as True.
any(s)	Returns True if any of the values in the iterable s evaluate as True.
ascii(x)	Creates a printable representation of the object x just like the repr(), but only uses ASCII characters in the result. Non-ASCII characters are turned into appropriate escape sequences. This can be used to view Unicode strings in a terminal or shell that doesn't support Unicode.
bin(x)	Returns a string containing the binary representation of the integer x.
bool([x])	Type representing Boolean values True and False. If used to convert x, it returns True if x evaluates to true using the usual truth-testing semantics (that is, nonzero number, non-empty list, and so on). Otherwise, False is returned. False is also the default value returned if bool() is called without any arguments. The bool class inherits from int so the Boolean values True and False can be used as integers with values 1 and 0 in mathematical calculations.
bytearray([x]), bytes([x])	A type representing a mutable array of bytes. Both array and string interfaces are supported, but use e.g a.split(b',') not a.split(',').
bytearray(s ,encoding)	An alternative calling convention for creating a bytearray instance from characters in a string s where encoding specifies the character encoding to use in the conversion.
bytes(s, encoding)	An alternative calling convention for creating a bytes instance from characters in a string s where encoding specifies the character encoding to use.
chr(x)	Converts an integer value, x, into a one-character string. x must represent a valid Unicode code point. If x is out of range, a ValueError exception is raised.
classmethod(func)	This function creates a class method for the function func. It is typically only used inside class definitions where it is implicitly invoked by the @classmethod decorator.
cmp(x, y)	Compares x and y and returns a negative number if x < y, a positive number if x > y, or 0 if x == y. Any two objects can be compared, although the result may be meaningless if the two objects have no meaningful comparison method. In certain circumstances, such comparisons may also raise an exception.
compile(string, filename, kind [, flags [, dont_inherit]])	Compiles string into a code object for use with exec() or eval().
complex([real [, imag]])	Type representing a complex number with real and imaginary components, real and imag, which can be supplied as any numeric type. If imag is omitted, the imaginary component is set to zero. If real is passed as a string, the string is parsed and converted to a complex number. In this case, imag should be omitted. If no arguments are given, 0j is returned.
delattr(object, attr)	Deletes an attribute of an object. attr is a string. Same as del object.attr.
dict([m]) or **dict(key1 = value1, key2 = value2, ...)**	Type representing a dictionary.

dir([object])	Returns a sorted list of attribute names. If object is a module, it contains the list of symbols defined in that module. If object is a type or class object, it returns a list of attribute names.
divmod(a, b)	Returns the quotient and remainder of long division as a tuple. For integers, the value (a // b, a % b) is returned. For floats, (math.floor(a / b), a % b) is returned. This function may not be called with complex numbers.
enumerate(iter[, initial value)	Given an iterable object, iter, returns a new iterator (of type enumerate) that produces tuples containing a count and the value produced from iter. For example, if iter produces a, b, c, then enumerate(iter) produces (0,a), (1,b), (2,c).
eval(expr [, globals [, locals]])	Evaluates an expression.
exec(code [, global [, locals]])	Executes Python statements.
filter(function, iterable)	This creates an iterator that iterates through the objects from iterable for which function evaluates to true. If function is None, the identity function is used and all the elements of iterable that are false are removed.
float([x])	Type representing a floating-point number. If x is a number, it is converted to a float. If x is a string, it is parsed into a float. If no argument is supplied, 0.0 is returned.
format(value [, format_spec])	* Converts value to a formatted string according to the format specification string in format_spec.
frozenset([items])	* Type representing an immutable set object populated with values taken from items that must be an iterable. The values must also be immutable. If no argument is given, an empty set is returned.
getattr(object, name [,default])	* Returns the value of a named attribute of an object. name is a string containing the attribute name. default is an optional value to return if no such attribute exists. Otherwise, AttributeError is raised. Same as object.name.
globals()	* Returns the dictionary of the current module that represents the global namespace. When called inside another function or method, it returns the global namespace of the module in which the function or method was defined.
hasattr(object, name)	* Returns True if name is the name of an attribute of object. False is returned otherwise. name is a string.
hash(object)	* Returns an integer hash value for an object (if possible). The hash value is primarily used in the implementation of dictionaries, sets, and other mapping objects. The hash value is the same for any two objects that compare as equals. Mutable objects don't define a hash value, although user-defined classes can define a method _ _hash_ _() to support this operation.
help([object])	Calls the built-in help system during interactive sessions. object may be a string representing the name of a module, class, function, method, keyword, or documentation topic. If it is any other kind of object, a help screen related to that object will be produced. If no argument is supplied, an interactive help tool will start and provide more

	information.
hex(x)	Creates a hexadecimal string from an integer x.
id(object)	* Returns the unique integer identity of object. You should not interpret the return value in any way (that is, as a memory location).
input([prompt])	A prompt is printed to standard output and a single line of input is read without any kind of evaluation or modification.
int(x [,base])	Type representing an integer. If x is a number, it is converted to an integer by truncating toward 0. If it is a string, it is parsed into an integer value. base optionally specifies a base when converting from a string.
isinstance(object, classobj)	* Returns True if object is an instance of classobj, is a subclass of classobj, or belongs to an abstract base class classobj.The classobj parameter can also be a tuple of possible types or classes. For example, isinstance(s, (list,tuple)) returns True if s is a tuple or a list.
issubclass(class1, class2)	* Returns True if class1 is a subclass of (derived from) class2 or if class1 is registered with an abstract base class class2. class2 can also be a tuple of possible classes, in which case each class will be checked. Note that issubclass(A, A) is true.
iter(object [,sentinel])	* Returns an iterator for producing items in object. A TypeError will be generated if object does not support iteration.
len(s)	Returns the number of items contained in s. s should be a list, tuple, string, set, or dictionary. A TypeError is generated if s is an iterable such as a generator.
list([items])	Type representing a list. items may be any iterable object, the values of which are used to populate the list. If items is already a list, a copy is made. If no argument is given, an empty list is returned.
locals()	* Returns a dictionary corresponding to the local namespace of the caller. This dictionary should only be used to inspect the execution environment—it is not safe to modify the contents of this dictionary.
map(function, items, ...)	* An iterator that applies function to every item of items is created. If multiple input sequences are supplied, function is assumed to take that many arguments, with each argument taken from a different sequence. When processing multiple input sequences, the result is only as long as the shortest sequence.
max(s [, args, ...])	For a single argument, s, this function returns the maximum value of the items in s, which may be any iterable object. For multiple arguments, it returns the largest of the arguments.
min(s [, args, ...])	For a single argument, s, this function returns the minimum value of the items in s, which may be any iterable object. For multiple arguments, it returns the smallest of the arguments.
next(s [, default])	* Returns the next item from the iterator s. If the iterator has no more items, a StopIteration exception is raised unless a value is supplied to the default argument. In that case, default is returned instead. For portability, you should always use this function instead of calling s.next() directly on an iterator s.
object()	The base class for all objects in Python. You can call it to create an instance, but the result isn't especially interesting.
oct(x)	Converts an integer, x, to an octal string.

open(filename [, mode [, bufsize [, encoding [, errors [, newline [,closefd]]]]]])	This opens the file filename and returns a stream. The actual type depends on whether you opened the file in binary (BufferedReader, BufferedWriter, BufferedRandom) or text mode (TextIOWrapper).
ord(c)	Returns the integer ordinal value of a single character, c. For ordinary characters, a value in the range [0,255] is returned. For single Unicode characters, a value in the range [0,65535] is usually returned. Character c may also be a Unicode surrogate pair, in which case it is converted into the appropriate Unicode code point.
pow(x, y [, z])	Returns x ** y. If z is supplied, this function returns (x ** y) % z. If all three arguments are given, they must be integers and y must be nonnegative.
print(value, ... [, sep=separator, end=ending, file=outfile])	Function for printing a series of values. As input, you can supply any number of values, all of which are printed on the same line. The sep keyword argument is used to specify a different separator character (a space by default). The end keyword argument specifies a different line ending ('\n' by default).The file keyword argument redirects the output to a file object.
property([fget [,fset [,fdel [,doc]]]])	* Creates a property attribute for classes. fget is a function that returns the attribute value, fset sets the attribute value, and fdel deletes an attribute. doc provides a documentation string. These parameters may be supplied using keyword arguments—for example, property(fget=getX, doc="some text").
range([start,] stop) or **range(start, stop, step)**	This creates a special range object that computes its values on demand from start to stop. Step indicates a stride and is set to 1 if omitted. If start is omitted (when range() is called with one argument), it defaults to 0. A negative step creates a list of numbers in descending order.
repr(object)	* Returns a string representation of object. In most cases, the returned string is an expression that can be passed to eval() to re-create the object.
reversed(s)	Creates a reverse iterator for sequence s. This function only works if s implements the sequence methods _ _len_ _() and _ _getitem_ _(). In addition, s must index items starting at 0. It does not work with generators or iterators.
round(x [, n])	Rounds the result of rounding the floating-point number x to the closest multiple of 10 to the power minus n. If n is omitted, it defaults to 0. If two multiples are equally close, rounds toward 0 if the previous digit is even and away from 0 otherwise (for example, 0.5 is rounded to 0.0 and 1.5 is rounded to 2).
set([items])	* Creates a set populated with items taken from the iterable object items.The items must be immutable. If items contains other sets, those sets must be of type frozenset. If items is omitted, an empty set is returned.
setattr(object, name, value)	* Sets an attribute of an object. name is a string. Same as object.name = value.
slice([start,] stop) or **slice(start, stop, step)**	Returns a slice object representing integers in the specified range. Slice objects are also generated by the extended slice syntax a[i:i:k].
sorted(iterable [, key=keyfunc [,	Creates a sorted list from items in iterable. The keyword argument key is a single argument function that transforms values before they are

reverse=reverseflag]])	passed to the compare function. The keyword argument reverse is a Boolean flag that specifies whether or not the resulting list is sorted in reverse order. The key and reverse arguments must be specified using keywords—for example, sorted(a,key=get_name).
staticmethod(func)	* Creates a static method for use in classes.This function is implicitly invoked by the @staticmethod decorator.
str([object])	Type representing a string. Strings are Unicode. If object is supplied, a string representation of its value is created by calling its _ _str_ _() method. This is the same string that you see when you print the object. If no argument is given, an empty string is created.
sum(items [,initial])	Computes the sum of a sequence of items taken from the iterable object items. initial provides the starting value and defaults to 0.This function only works with numbers.
super(type [, object])	* Returns an object that represents the superclasses of type. When used in a method with no arguments, type is set to the class in which the method is defined and object is set to the first argument of the method.
tuple([items])	Type representing a tuple. If supplied, items is an iterable object that is used to populate the tuple. However, if items is already a tuple, it's simply returned unmodified. If no argument is given, an empty tuple is returned.
type(object)	The base class of all types in Python. When called as a function, returns the type of object. This type is the same as the object's class. For common types such as integers, floats, and lists, the type will refer to one of the other built-in classes such as int, float, list, and so forth. For user-defined objects, the type is the associated class. For objects related to Python's internals, you will typically get a reference to one of the classes defined in the types module.
type(name,bases,dict)	* Creates a new type object (which is the same as defining a new class). name is the name of the type, bases is a tuple of base classes, and dict is a dictionary containing definitions corresponding to a class body.
vars([object])	* Returns the symbol table of object (usually found in its _ _dict_ _ attribute). If no argument is given, a dictionary corresponding to the local namespace is returned. The dictionary returned by this function should be assumed to be read-only. It's not safe to modify its contents.
zip([s1 [, s2 [,..]]])	Returns an iterator that produces a sequence of tuples where the *n*th tuple is (s1[n], s2[n], ...). The resulting list is truncated to the length of the shortest argument sequence. If no arguments are given, an empty list is returned.

Exception

Exception Base Classes

BaseException	The root class for all exceptions. All built-in exceptions are derived from this class.
Exception	The base class for all program-related exceptions that includes all built-in exceptions except for SystemExit, GeneratorExit, and KeyboardInterrupt. User-defined exceptions should be defined by inheriting from Exception.
ArithmeticError	The base class for arithmetic exceptions, including OverflowError, ZeroDivisionError, and FloatingPointError.
LookupError	The base class for indexing and key errors, including IndexError and KeyError.
EnvironmentError	The base class for errors that occur outside Python, including IOError and OSError.

Predefined Exception Classes

AssertionError	Failed assert statement.
AttributeError	Failed attribute reference or assignment.
EOFError	End of file. Generated by the built-in functions input() and raw_input(). It should be noted that most other I/O operations such as the read() and readline() methods of files return an empty string to signal EOF instead of raising an exception.
FloatingPointError	Failed floating-point operation. It should be noted that floating-point exception handling is a tricky problem and only that this exception only gets raised if Python has been configured and built in a way that enables it. It is more common for floating-point errors to silently produce results such as float('nan') or float('inf'). A subclass of ArithmeticError.
GeneratorExit	Raised inside a generator function to signal termination. This happens when a generator is destroyed prematurely (before all generator values are consumed) or the close() method of a generator is called. If a generator ignores this exception, the generator is terminated and the exception is silently ignored.
IOError	Failed I/O operation. The value is an IOError instance with the attributes errno, strerror, and filename. errno is an integer error number, strerror is a string error message, and filename is an optional filename. A subclass of EnvironmentError.
ImportError	Raised when an import statement can’t find a module or when from can’t find a name in a module.
IndentationError	Indentation error. A subclass of SyntaxError.
IndexError	Sequence subscript out of range. A subclass of LookupError.
KeyError	Key not found in a mapping. A subclass of LookupError.
KeyboardInterrupt	Raised when the user hits the interrupt key (usually Ctrl+C).
MemoryError	Recoverable out-of-memory error.
NameError	Name not found in local or global namespaces.
NotImplementedError	Unimplemented feature. Can be raised by base classes that require derived classes to implement certain methods. A subclass of RuntimeError.
OSError	Operating system error. Primarily raised by functions in the os module. The value is the same as for IOError. A subclass of EnvironmentError.

OverflowError	Result of an integer value being too large to be represented.This exception usually only arises if large integer values are passed to objects that internally rely upon fixed precision machine integers in their implementation. For example, this error can arise with range objects if you specify starting or ending values that exceed 32 bits in size. A subclass of ArithmeticError.
ReferenceError	Result of accessing a weak reference after the underlying object has been destroyed. See the weakref module.
RuntimeError	A generic error not covered by any of the other categories.
StopIteration	Raised to signal the end of iteration.This normally happens in the next() method of an object or in a generator function.
SyntaxError	Parser syntax error. Instances have the attributes filename, lineno, offset, and text, which can be used to gather more information.
SystemError	Internal error in the interpreter. The value is a string indicating the problem.
SystemExit	Raised by the sys.exit() function. The value is an integer indicating the return code. If it's necessary to exit immediately, os._exit() can be used.
TabError	Inconsistent tab usage. Generated when Python is run with the -tt option. A subclass of SyntaxError.
TypeError	Occurs when an operation or a function is applied to an object of an inappropriate type.
UnboundLocalError	Unbound local variable referenced. This error occurs if a variable is referenced before it's defined in a function. A subclass of NameError.
UnicodeError	Unicode encoding or decoding error. A subclass of ValueError.
UnicodeEncodeError	Unicode encoding error. A subclass of UnicodeError.
UnicodeDecodeError	Unicode decoding error. A subclass of UnicodeError.
UnicodeTranslateError	Unicode error occurred during translation. A subclass of UnicodeError.
ValueError	Generated when the argument to a function or an operation is the right type but an inappropriate value.
WindowsError	Generated by failed system calls on Windows. A subclass of OSError.
ZeroDivisionError	Dividing by zero. A subclass of ArithmeticError.

math module functions

Function	**Description**
acos(x)	Returns the arc cosine of x.
acosh(x)	Returns the hyperbolic arc cosine of x.
asin(x)	Returns the arc sine of x.
asinh(x)	Returns the hyperbolic arc sine of x.
atan(x)	Returns the arc tangent of x.
atan2(y, x)	Returns atan(y / x).
atanh(x)	Returns the hyperbolic arc tangent of x.
ceil(x)	Returns the ceiling of x.
copysign(x,y)	Returns x with the same sign as y.
cos(x)	Returns the cosine of x.
cosh(x)	Returns the hyperbolic cosine of x.
degrees(x)	Converts x from radians to degrees.
radians(x)	Converts x from degrees to radians.
exp(x)	Returns e ** x.
fabs(x)	Returns the absolute value of x.
factorial(x)	Returns x factorial.
floor(x)	Returns the floor of x.
fmod(x, y)	Returns x % y as computed by the C fmod() function.
frexp(x)	Returns the positive mantissa and exponent of x as a tuple.
fsum(s)	Returns the full precision sum of floating-point values in the iterable sequence s.
hypot(x, y)	Returns the Euclidean distance, sqrt(x * x + y * y).
isinf(x)	Return True if x is infinity.
isnan(x)	Returns True if x is NaN.
ldexp(x, i)	Returns x * (2 ** i).
log(x [, base])	Returns the logarithm of x to the given base. If base is omitted, this function computes the natural logarithm.
log10(x)	Returns the base 10 logarithm of x.
log1p(x)	Returns the natural logarithm of 1+x.
modf(x)	Returns the fractional and integer parts of x as a tuple. Both have the same sign as x.
pow(x, y)	Returns x ** y.
sin(x)	Returns the sine of x.
sinh(x)	Returns the hyperbolic sine of x.
sqrt(x)	Returns the square root of x.
tan(x)	Returns the tangent of x.
tanh(x)	Returns the hyperbolic tangent of x.
trunc(x)	Truncates x to the nearest integer towards 0.

Constants

Constant	**Description**
pi	Mathematical constant pi
e	Mathematical constant e