Python Quick Reference

Contents

Numbers

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

Number Type Conversion:

Python converts numbers internally in an expression containing mixed types to a common type for evaluation. But sometimes, you'll need to coerce a number explicitly from one type to another to satisfy the requirements of an operator or function parameter.

Type int(x) to convert x to a plain integer.
Type long(x) to convert x to a long integer.
Type float(x) to convert x to a floating-point number.
Type complex(x) to convert x to a complex number with real part x and imaginary part zero.
Type complex(x, y) to convert x and y to a complex number with real part x and imaginary part y. x and y are numeric expressions

Built-in Number Functions:

Mathematical Functions:

Python includes following functions that perform mathematical calculations.

Function	Returns (description)
abs(x)	The absolute value of x: the (positive) distance between x and zero.
ceil(x)	The ceiling of x: the smallest integer not less than x

cmp(x, y)	-1 if x < y, 0 if x == y, or 1 if x > y
exp(x)	The exponential of x: e^x
fabs(x)	The absolute value of x.
floor(x)	The floor of x: the largest integer not greater than x
log(x)	The natural logarithm of x, for x> 0
log10(x)	The base-10 logarithm of x for x> 0 .
max(x1, x2,...)	The largest of its arguments: the value closest to positive infinity
min(x1, x2,...)	The smallest of its arguments: the value closest to negative infinity
modf(x)	The fractional and integer parts of x in a two-item tuple. Both parts have the same sign as x. The integer part is returned as a float.
pow(x, y)	The value of x**y.
round(x [,n])	x rounded to n digits from the decimal point. Python rounds away from zero as a tie-breaker: round(0.5) is 1.0 and round(-0.5) is -1.0.
sqrt(x)	The square root of x for x > 0

Random Number Functions:

Random numbers are used for games, simulations, testing, security, and privacy applications. Python includes following functions that are commonly used.

Function	Returns (description)
choice(seq)	A random item from a list, tuple, or string.
randrange ([start,] stop [,step])	A randomly selected element from range(start, stop, step)

random()	A random float r, such that 0 is less than or equal to r and r is less than 1
seed([x])	Sets the integer starting value used in generating random numbers. Call this function before calling any other random module function. Returns None.
shuffle(lst)	Randomizes the items of a list in place. Returns None.
uniform(x, y)	A random float r, such that x is less than or equal to r and r is less than y

Trigonometric Functions:

Python includes following functions that perform trigonometric calculations.

Function	Description
acos(x)	Return the arc cosine of x, in radians.
asin(x)	Return the arc sine of x, in radians.
atan(x)	Return the arc tangent of x, in radians.
atan2(y, x)	Return atan(y / x), in radians.
cos(x)	Return the cosine of x radians.
hypot(x, y)	Return the Euclidean norm, sqrt(x*x + y*y).
sin(x)	Return the sine of x radians.
tan(x)	Return the tangent of x radians.
degrees(x)	Converts angle x from radians to degrees.
radians(x)	Converts angle x from degrees to radians.

Mathematical Constants:

The module also defines two mathematical constants:

Constant	Description
pi	The mathematical constant pi.
e	The mathematical constant e.

What is an operator?

Simple answer can be given using expression 4 + 5 is equal to 9. Here 4 and 5 are called operands and + is called operator. Python language supports following type of operators.

Arithmetic Operators
Comparison Operators
Logical (or Relational) Operators
Assignment Operators
Conditional (or ternary) Operators

Lets have a look on all operators one by one.

Python Arithmetic Operators:

Assume variable a holds 10 and variable b holds 20 then:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right	b % a will give 0

	hand operand and returns remainder	
**	Exponent - Performs exponential (power) calculation on operators	a**b will give 10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 is equal to 4 and 9.0//2.0 is equal to 4.0

Python Comparison Operators:

Assume variable a holds 10 and variable b holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(a == b) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.

Python Assignment Operators:

Assume variable a holds 10 and variable b holds 20 then:

Operator	Description	Example
=	Simple assignment operator, Assigns	c = a + b will assigne value of a + b into c

	values from right side operands to left side operand	
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / a
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**=	Exponent AND assignment operator, Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//=	Floor Dividion and assigns a value, Performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Python Bitwise Operators:

Bitwise operator works on bits and perform bit by bit operation.

Assume if a = 60; and b = 13; Now in binary format they will be as follows:

a = 0011 1100

b = 0000 1101

---

a&b = 0000 1000

alb = 0011 1101

a^b = 0011 0001

~a = 1100 0011

There are following Bitwise operators supported by Python language

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(a & b) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in eather operand.	(a | b) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(a ^ b) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the efect of 'flipping' bits.	(~a) will give -60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	a << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 will give 15 which is 0000 1111

Python Logical Operators:

There are following logical operators supported by Python language

Assume variable a holds 10 and variable b holds 20 then:

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(a and b) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(a or b) is true.
not	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	not(a && b) is false.

Python Membership Operators:

In addition to the operators discussed previously, Python has membership operators, which test for membership in a sequence, such as strings, lists, or tuples.

There are two membership operators explained below:

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is a member of sequence y.

Python Identity Operators:

Identity operators compare the memory locations of two objects.

There are two Identity operators explained below:

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

Python Operators Precedence

The following table lists all operators from highest precedence to lowest.

Operator	Description
**	Exponentiation (raise to the power)
~+-	Ccomplement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+-	Addition and subtraction

>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^|	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += |= &= >>= <<= *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
note or and	Logical operators

Reserved Words:

The following list shows the reserved words in Python. These reserved words may not be used as constant or variable or any other identifier names.

Keywords contain lowercase letters only.

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Strings

Here is the list of complete set of symbols which can be used along with %:

Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPERcase 'E')
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

Other supported symbols and functionality are listed in the following table:

Symbol	Functionality
*	argument specifies width or precision
-	left justification
+	display the sign
<sp>	leave a blank space before a positive number
#	add the octal leading zero ('0') or hexadecimal leading '0x' or '0X', depending on whether 'x' or 'X' were used.
0	pad from left with zeros (instead of spaces)
%	'%%' leaves you with a single literal '%'
(var)	mapping variable (dictionary arguments)

m.n.	m is the minimum total width and n is the number of digits to display after the decimal point (if appl.)

Built-in String Methods:

Python includes following string method:

SN	Methods with Description
1	capitalize() Capitalizes first letter of string
2	center(width, fillchar) Returns a space-padded string with the original string centered to a total of width columns
3	count(str, beg= 0,end=len(string)) Counts how many times str occurs in string, or in a substring of string if starting index beg and ending index end are given
3	decode(encoding='UTF-8',errors='strict') Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.
4	encode(encoding='UTF-8',errors='strict') Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'.
5	endswith(suffix, beg=0, end=len(string)) Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; Returns true if so, and false otherwise
6	expandtabs(tabsize=8) Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided
7	find(str, beg=0 end=len(string)) Determine if str occurs in string, or in a substring of string if starting index beg and ending index end are given; returns index if found and -1 otherwise
8	index(str, beg=0, end=len(string)) Same as find(), but raises an exception if str not found
9	isa1num() Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise
10	isalpha() Returns true if string has at least 1 character and all characters are alphabetic and false otherwise
11	isdigit() Returns true if string contains only digits and false otherwise
12	islower() Returns true if string has at least 1 cased character and all cased characters are in lowercase and false

	otherwise
13	isnumeric() Returns true if string contains only numeric characters and false otherwise
14	isspace() Returns true if string contains only whitespace characters and false otherwise
15	istitle() Returns true if string is properly "titlecased" and false otherwise
16	isupper() Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise
17	join(seq) Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string
18	len(string) Returns the length of the string
19	ljust(width[, fillchar]) Returns a space-padded string with the original string left-justified to a total of width columns
20	lower() Converts all uppercase letters in string to lowercase
21	lstrip() Removes all leading whitespace in string
22	maketrans() Returns a translation table to be used in translate function.
23	max(str) Returns the max alphabetical character from the string str
24	min(str) Returns the min alphabetical character from the string str
25	replace(old, new [, max]) Replaces all occurrences of old in string with new, or at most max occurrences if max given
26	rfind(str, beg=0,end=len(string)) Same as find(), but search backwards in string
27	rindex(str, beg=0, end=len(string)) Same as index(), but search backwards in string
28	rjust(width,[, fillchar]) Returns a space-padded string with the original string right-justified to a total of width columns.
29	rstrip() Removes all trailing whitespace of string

30	split(str="", num=string.count(str)) Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given
31	splitlines(num=string.count('\n')) Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed
32	startswith(str, beg=0,end=len(string)) Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring str; Returns true if so, and false otherwise
33	strip([chars]) Performs both lstrip() and rstrip() on string
34	swapcase() Inverts case for all letters in string
35	title() Returns "titlecased" version of string, that is, all words begin with uppercase, and the rest are lowercase
36	translate(table, deletechars="") Translates string according to translation table str(256 chars), removing those in the del string
37	upper() Converts lowercase letters in string to uppercase
38	zfill (width) Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero)

Basic List Operations:

Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.

In fact, lists respond to all of the general sequence operations we used on strings in the prior chapter :

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	123	Iteration

Indexing, Slicing, and Matrixes:

Because lists are sequences, indexing and slicing work the same way for lists as they do for strings.

Assuming following input:

```
L = ['spam', 'Spam', 'SPAM!']
```

Python Expression	Results	Description
L[2]	'SPAM!'	Offsets start at zero
L[-2]	'Spam'	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

Built-in List Functions & Methods:

Python includes following list functions

SN	Function with Description
1	cmp(list1, list2) Compares elements of both lists.
2	len(list) Gives the total length of the list.
3	max(list) Returns item from the list with max value.
4	min(list) Returns item from the list with min value.
5	list(seq) Converts a tuple into list.

Python includes following list methods

SN	Methods with Description
1	list.append(obj) Appends object obj to list
2	list.count(obj) Returns count of how many times obj occurs in list
3	list.extend(seq) Appends the contents of seq to list
4	list.index(obj) Returns the lowest index in list that obj appears
5	list.insert(index, obj) Inserts object obj into list at offset index
6	list.pop(obj=list[-1]) Removes and returns last object or obj from list
7	list.remove(obj) Removes object obj from list
8	list.reverse() Reverses objects of list in place
9	list.sort([func]) Sorts objects of list, use compare func if given

Built-in Tuple Functions:

Python includes following tuple functions

SN	Function with Description
1	cmp(tuple1, tuple2) Compares elements of both tuples.
2	len(tuple) Gives the total length of the tuple.
3	max(tuple) Returns item from the tuple with max value.
4	min(tuple) Returns item from the tuple with min value.
5	tuple(seq) Converts a list into tuple.

Built-in Dictionary Functions & Methods:

Python includes following dictionary functions

SN	Function with Description
1	cmp(dict1, dict2) Compares elements of both dict.
2	len(dict) Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
3	str(dict) Produces a printable string representation of a dictionary
4	type(variable) Returns the type of the passed variable. If passed variable is dictionary then it would return a dictionary type.

Python includes following dictionary methods

SN	Methods with Description
1	dict.clear() Removes all elements of dictionary dict
2	dict.copy() Returns a shallow copy of dictionary dict
2	dict.fromkeys() Create a new dictionary with keys from seq and values set to value.
3	dict.get(key, default=None) For key key, returns value or default if key not in dictionary
4	dict.has_key(key) Returns true if key in dictionary dict, false otherwise
5	dict.items() Returns a list of dict's (key, value) tuple pairs
6	dict.keys() Returns list of dictionary dict's keys
7	dict.setdefault(key, default=None) Similar to get(), but will set dict[key]=default if key is not already in dict
8	dict.update(dict2) Adds dictionary dict2's key-values pairs to dict
9	dict.values() Returns list of dictionary dict2's values

What is TimeTuple?

Many of Python's time functions handle time as a tuple of 9 numbers, as shown below:

Index	Field	Values
0	4-digit year	2008
1	Month	1 to 12
2	Day	1 to 31
3	Hour	0 to 23
4	Minute	0 to 59
5	Second	0 to 61 (60 or 61 are leap-seconds)
6	Day of Week	0 to 6 (0 is Monday)
7	Day of year	1 to 366 (Julian day)
8	Daylight savings	-1, 0, 1, -1 means library determines DST

The above tuple is equivalent to struct_time structure. This structure has following attributes:

Index	Attributes	Values
0	tm_year	2008
1	tm_mon	1 to 12
2	tm_mday	1 to 31
3	tm_hour	0 to 23

4	tm_min	0 to 59
5	tm_sec	0 to 61 (60 or 61 are leap-seconds)
6	tm_wday	0 to 6 (0 is Monday)
7	tm_yday	1 to 366 (Julian day)
8	tm_isdst	-1, 0, 1, -1 means library determines DST

The time Module:

There is a popular time module available in Python which provides functions for working with times, and for converting between representations. Here is the list of all available methods:

SN	Function with Description
1	time.altzone The offset of the local DST timezone, in seconds west of UTC, if one is defined. This is negative if the local DST timezone is east of UTC (as in Western Europe, including the UK). Only use this if daylight is nonzero.
2	time.asctime([tupletime]) Accepts a time-tuple and returns a readable 24-character string such as 'Tue Dec 11 18:07:14 2008'.
3	time.clock( ) Returns the current CPU time as a floating-point number of seconds. To measure computational costs of different approaches, the value of time.clock is more useful than that of time.time().
4	time.ctime([secs]) Like asctime(localtime(secs)) and without arguments is like asctime( )
5	time.gmtime([secs]) Accepts an instant expressed in seconds since the epoch and returns a time-tuple t with the UTC time. Note : t.tm_isdst is always 0
6	time.localtime([secs]) Accepts an instant expressed in seconds since the epoch and returns a time-tuple t with the local time (t.tm_isdst is 0 or 1, depending on whether DST applies to instant secs by local rules).
7	time.mktime(tupletime) Accepts an instant expressed as a time-tuple in local time and returns a floating-point value with the instant expressed in seconds since the epoch.
8	time.sleep(secs) Suspends the calling thread for secs seconds.

9	time.strftime(fmt[,tupletime]) Accepts an instant expressed as a time-tuple in local time and returns a string representing the instant as specified by string fmt.
10	time.strptime(str,fmt='%a %b %d %H:%M:%S %Y') Parses str according to format string fmt and returns the instant in time-tuple format.
11	time.time( ) Returns the current time instant, a floating-point number of seconds since the epoch.
12	time.tzset() Resets the time conversion rules used by the library routines. The environment variable TZ specifies how this is done.

There are following two important attributes available with time module:

SN	Attribute with Description
1	time.timezone Attribute time.timezone is the offset in seconds of the local time zone (without DST) from UTC (>0 in the Americas; <=0 in most of Europe, Asia, Africa).
2	time.tzname Attribute time.tzname is a pair of locale-dependent strings, which are the names of the local time zone without and with DST, respectively.

The calendar Module

The calendar module supplies calendar-related functions, including functions to print a text calendar for a given month or year.

By default, calendar takes Monday as the first day of the week and Sunday as the last one. To change this, call calendar.setfirstweekday() function.

Here is a list of functions available with the calendar module:

SN	Function with Description
1	calendar.calendar(year,w=2,l=1,c=6) Returns a multiline string with a calendar for year year formatted into three columns separated by c spaces. w is the width in characters of each date; each line has length 21*w+18+2*c. l is the number

	of lines for each week.
2	calendar.firstweekday( ) Returns the current setting for the weekday that starts each week. By default, when calendar is first imported, this is 0, meaning Monday.
3	calendar.isleap(year) Returns True if year is a leap year; otherwise, False.
4	calendar.leapdays(y1,y2) Returns the total number of leap days in the years within range(y1,y2).
5	calendar.month(year,month,w=2,l=1) Returns a multiline string with a calendar for month month of year year, one line per week plus two header lines. w is the width in characters of each date; each line has length 7*w+6. l is the number of lines for each week.
6	calendar.monthcalendar(year,month) Returns a list of lists of ints. Each sublist denotes a week. Days outside month month of year year are set to 0; days within the month are set to their day-of-month, 1 and up.
7	calendar.monthrange(year,month) Returns two integers. The first one is the code of the weekday for the first day of the month month in year year; the second one is the number of days in the month. Weekday codes are 0 (Monday) to 6 (Sunday); month numbers are 1 to 12.
8	calendar.prcal(year,w=2,l=1,c=6) Like print calendar.calendar(year,w,l,c).
9	calendar.prmonth(year,month,w=2,l=1) Like print calendar.month(year,month,w,l).
10	calendar.setfirstweekday(weekday) Sets the first day of each week to weekday code weekday. Weekday codes are 0 (Monday) to 6 (Sunday).
11	calendar.timegm(tupletime) The inverse of time.gmtime: accepts a time instant in time-tuple form and returns the same instant as a floating-point number of seconds since the epoch.
12	calendar.weekday(year,month,day) Returns the weekday code for the given date. Weekday codes are 0 (Monday) to 6 (Sunday); month numbers are 1 (January) to 12 (December).

File Operations

Here is a list of the different modes of opening a file:

Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer will be at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer will be at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

The file object atrributes:

Once a file is opened and you have one file object, you can get various information related to that file.

Here is a list of all attributes related to file object:

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.

SN	Methods with Description
1	file.close() Close the file. A closed file cannot be read or written any more.
2	file.flush() Flush the internal buffer, like stdio's fflush. This may be a no-op on some file-like objects.
3	file.fileno() Return the integer file descriptor that is used by the underlying implementation to request I/O operations from the operating system.
4	file.isatty() Return True if the file is connected to a tty(-like) device, else False.
5	file.next() Returns the next line from the file each time it is being called.
6	file.read([size]) Read at most size bytes from the file (less if the read hits EOF before obtaining size bytes).
7	file.readline([size]) Read one entire line from the file. A trailing newline character is kept in the string.
8	file.readlines([sizehint]) Read until EOF using readline() and return a list containing the lines. If the optional sizehint argument is present, instead of reading up to EOF, whole lines totalling approximately sizehint bytes (possibly after rounding up to an internal buffer size) are read.
9	file.seek(offset[, whence]) Set the file's current position.
10	file.tell() Return the file's current position
11	file.truncate([size]) Truncate the file's size. If the optional size argument is present, the file is truncated to (at most) that size.
12	file.write(str) Write a string to the file. There is no return value.
13	file.writelines(sequence) Write a sequence of strings to the file. The sequence can be any iterable object producing strings, typically a list of strings.

Regular Expressions

The match Function

This function attempts to match RE pattern to string with optional flags.

Here is the syntax for this function:

```
re.match(pattern, string, flags=0)
```

Here is the description of the parameters:

Parameter	Description
pattern	This is the regular expression to be matched.
string	This is the string which would be searched to match the pattern
flags	You can specifiy different flags using exclusive OR (|). These are modifiers which are listed in the table below.

The re.match function returns a match object on success, None on failure. We would use group(num) or groups() function of match object to get matched expression.

Match Object Methods	Description
group(num=0)	This methods returns entire match (or specific subgroup num)
groups()	This method return all matching subgroups in a tuple (empty if there weren't any)

The search Function

This function search for first occurrence of RE pattern within string with optional flags.

Here is the syntax for this function:

```
re.string(pattern, string, flags=0)
```

Here is the description of the parameters:

Parameter	Description
pattern	This is the regular expression to be matched.
string	This is the string which would be searched to match the pattern
flags	You can specifiy different flags using exclusive OR (|). These are modifiers which are listed in the table below.

Regular-expression Modifiers - Option Flags

Regular expression literals may include an optional modifier to control various aspects of matching. The modifier are specified as an optional flag. You can provide multiple modified using exclusive OR (|), as shown previously and may be represented by one of these:

Modifier	Description
re.I	Performs case-insensitive matching.
re.L	Interprets words according to the current locale.This interpretation affects the alphabetic group (\w and \W), as well as word boundary behavior (\b and \B).
re.M	Makes $ match the end of a line (not just the end of the string) and makes ^ match the start of any line (not just the start of the string).
re.S	Makes a period (dot) match any character, including a newline.
re.U	Interprets letters according to the Unicode character set. This flag affects the behavior of \w, \W, \b, \B.
re.X	Permits "cuter" regular expression syntax. It ignores whitespace (except inside a set [] or when escaped by a backslash), and treats unescaped # as a comment marker.

Regular-expression patterns:

Except for control characters, (+ ? . * ^ $ ( ) [ ] { } | \), all characters match themselves. You can escape a control character by preceding it with a backslash.

Following table lists the regular expression syntax that is available in Python.

Pattern	Description
^	Matches beginning of line.
$	Matches end of line.

.	Matches any single character except newline. Using m option allows it to match newline as well.
[...]	Matches any single character in brackets.
[^...]	Matches any single character not in brackets
re*	Matches 0 or more occurrences of preceding expression.
re+	Matches 1 or more occurrence of preceding expression.
re?	Matches 0 or 1 occurrence of preceding expression.
re{ n}	Matches exactly n number of occurrences of preceding expression.
re{ n,}	Matches n or more occurrences of preceding expression.
re{ n, m}	Matches at least n and at most m occurrences of preceding expression.
a| b	Matches either a or b.
(re)	Groups regular expressions and remembers matched text.
(?imx)	Temporarily toggles on i, m, or x options within a regular expression. If in parentheses, only that area is affected.
(?-imx)	Temporarily toggles off i, m, or x options within a regular expression. If in parentheses, only that area is affected.
(?: re)	Groups regular expressions without remembering matched text.
(?imx: re)	Temporarily toggles on i, m, or x options within parentheses.
(?-imx: re)	Temporarily toggles off i, m, or x options within parentheses.
(?#...)	Comment.
(?= re)	Specifies position using a pattern. Doesn't have a range.
(?! re)	Specifies position using pattern negation. Doesn't have a range.
(?> re)	Matches independent pattern without backtracking.
\w	Matches word characters.
\W	Matches nonword characters.
\s	Matches whitespace. Equivalent to [\t\n\r\f].
\S	Matches nonwhitespace.

\d	Matches digits. Equivalent to [0-9].
\D	Matches nondigits.
\A	Matches beginning of string.
\Z	Matches end of string. If a newline exists, it matches just before newline.
\z	Matches end of string.
\G	Matches point where last match finished.
\b	Matches word boundaries when outside brackets. Matches backspace (0x08) when inside brackets.
\B	Matches nonword boundaries.
\n, \t, etc.	Matches newlines, carriage returns, tabs, etc.
\1...\9	Matches nth grouped subexpression.
\10	Matches nth grouped subexpression if it matched already. Otherwise refers to the octal representation of a character code.

Regular-expression Examples:

Literal characters:

Example	Description
python	Match "python".

Character classes:

Example	Description
[Pp]ython	Match "Python" or "python"
rub[ye]	Match "ruby" or "rube"
[aeiou]	Match any one lowercase vowel
[0-9]	Match any digit; same as [0123456789]
[a-z]	Match any lowercase ASCII letter
[A-Z]	Match any uppercase ASCII letter
[a-zA-Z0-9]	Match any of the above
[^aeiou]	Match anything other than a lowercase vowel
[^0-9]	Match anything other than a digit

Special Character Classes:

Example	Description
.	Match any character except newline
\d	Match a digit: [0-9]
\D	Match a nondigit: [^0-9]
\s	Match a whitespace character: [\t\r\n\f]
\S	Match nonwhitespace: [^ \t\r\n\f]
\w	Match a single word character: [A-Za-z0-9_]
\W	Match a nonword character: [^A-Za-z0-9_]

Repetition Cases:

Example	Description
ruby?	Match "rub" or "ruby": the y is optional
ruby*	Match "rub" plus 0 or more ys
ruby+	Match "rub" plus 1 or more ys
\d{3}	Match exactly 3 digits
\d{3,}	Match 3 or more digits
\d{3,5}	Match 3, 4, or 5 digits

Nongreedy repetition:

This matches the smallest number of repetitions:

Example	Description
<.*>	Greedy repetition: matches "<python>perl>"
<.*?>	Nongreedy: matches "<python>" in "<python>perl>"

Grouping with parentheses:

Example	Description
\D\d+	No group: + repeats \d
(\D\d)+	Grouped: + repeats \D\d pair
([Pp]ython(,)?)+	Match "Python", "Python, python, python", etc.

Backreferences:

This matches a previously matched group again:

Example	Description
([Pp])ython&\1ails	Match python&rails or Python&Rails
(['"])[^\1]*\1	Single or double-quoted string. \1 matches whatever the 1st group matched . \2 matches whatever the 2nd group matched, etc.

Alternatives:

Example	Description
python|perl	Match "python" or "perl"
rub(y|le))	Match "ruby" or "ruble"
Python(!+|\?)	"Python" followed by one or more ! or one ?

Anchors:

This need to specify match position

Example	Description
^Python	Match "Python" at the start of a string or internal line
Python$	Match "Python" at the end of a string or line
\APython	Match "Python" at the start of a string
Python\Z	Match "Python" at the end of a string
\bPython\b	Match "Python" at a word boundary
\brub\B	\B is nonword boundary: match "rub" in "rube" and "ruby" but not alone
Python(?=!)	Match "Python", if followed by an exclamation point
Python(?!!)	Match "Python", if not followed by an exclamation point

Special syntax with parentheses:

Example	Description
R(?#comment)	Matches "R". All the rest is a comment
R(?i)uby	Case-insensitive while matching "uby"
R(?i:uby)	Same as above
rub(?:y|le))	Group only without creating \1 backreference

Networking

Sockets have their own vocabulary:

Term	Description
domain	The family of protocols that will be used as the transport mechanism. These values are constants such as AF_INET, PF_INET, PF_UNIX, PF_X25, and so on.
type	The type of communications between the two endpoints, typically SOCK_STREAM for connection-oriented protocols and SOCK_DGRAM for connectionless protocols.
protocol	Typically zero, this may be used to identify a variant of a protocol within a domain and type.
hostname	The identifier of a network interface: A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation A string "<broadcast>", which specifies an INADDR_BROADCAST address. A zero-length string, which specifies INADDR_ANY, or An Integer, interpreted as a binary address in host byte order.
port	Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service.

The socket Module:

To create a socket, you must use the socket.socket() function available in socket module, which has the general syntax:

```
s = socket.socket (socket_family, socket_type, protocol=0)
```

Here is the description of the parameters:

socket_family: This is either AF_UNIX or AF_INET, as explained earlier.
socket_type: This is either SOCK_STREAM or SOCK_DGRAM.
protocol: This is usually left out, defaulting to 0.

Once you have socket object, then you can use required functions to create your client or server program. Following is the list of functions required:

Server Socket Methods:

Method	Description
s.bind()	This method binds address (hostname, port number pair) to socket.
s.listen()	This method sets up and start TCP listener.
s.accept()	This passively accept TCP client connection, waiting until connection arrives (blocking).

Client Socket Methods:

Method	Description
s.connect()	This method actively initiates TCP server connection.

General Socket Methods:

Method	Description
s.recv()	This method receives TCP message
s.send()	This method transmits TCP message
s.recvfrom()	This method receives UDP message
s.sendto()	This method transmits UDP message
s.close()	This method closes socket
socket.gethostname()	Returns the hostname.

A Simple Server

```
#!/usr/bin/python               # This is server.py file

import socket                   # Import socket module

s = socket.socket()             # Create a socket object
host = socket.gethostname()     # Get local machine name
port = 12345                    # Reserve a port for your service.
s.bind((host, port))            # Bind to the port


s.listen(5)# Now wait for client connection.
while True:
   c, addr = s.accept()# Establish connection with client.
   print 'Got connection from', addr
   c.send('Thank you for connecting')
   c.close()# Close the connection
```

A Simple Client

```
#!/usr/bin/python               # This is client.py file

import socket                   # Import socket module

s = socket.socket()# Create a socket object
host = socket.gethostname() # Get local machine name
port = 12345# Reserve a port for your service.


s.connect((host, port))
print s.recv(1024)
s.close
                                # Close the socket when done
```

Python Internet modules

A list of some important modules which could be used in Python Network/Internet programming.

Protocol	Common function	Port No	Python module
HTTP	Web pages	80	httplib, urllib, xmlrpclib
NNTP	Usenet news	119	nntplib
FTP	File transfers	20	ftplib, urllib
SMTP	Sending email	25	smtplib
POP3	Fetching email	110	poplib
IMAP4	Fetching email	143	imaplib
Telnet	Command lines	23	telnetlib
Gopher	Document transfers	70	gopherlib, urllib

Please check all the libraries mentioned above to work with FTP, SMTP, POP, and IMAP protocols.

Bibliography

The only website all this was taken from:

http://www.tutorialspoint.com/python/index.htm

…more File I/O

The **os** module provides a big range of useful methods to manipulate files and directories. Most of the useful methods are listed here:

SN	Methods with Description
1	os.access(path, mode) Use the real uid/gid to test for access to path.
2	os.chdir(path) Change the current working directory to path
3	os.chflags(path, flags) Set the flags of path to the numeric flags.
4	os.chmod(path, mode) Change the mode of path to the numeric mode.
5	os.chown(path, uid, gid) Change the owner and group id of path to the numeric uid and gid.
6	os.chroot(path) Change the root directory of the current process to path.
7	os.close(fd) Close file descriptor fd.
8	os.closerange(fd_low, fd_high) Close all file descriptors from fd_low (inclusive) to fd_high (exclusive), ignoring errors.
9	os.dup(fd) Return a duplicate of file descriptor fd.
10	os.dup2(fd, fd2) Duplicate file descriptor fd to fd2, closing the latter first if necessary.
11	os.fchdir(fd) Change the current working directory to the directory represented by the file descriptor fd.
12	os.fchmod(fd, mode) Change the mode of the file given by fd to the numeric mode.
13	os.fchown(fd, uid, gid) Change the owner and group id of the file given by fd to the numeric uid and gid.
14	os.fdatasync(fd) Force write of file with filedescriptor fd to disk.
15	os.fdopen(fd[, mode[, bufsize]]) Return an open file object connected to the file descriptor fd.
16	os.fpathconf(fd, name) Return system configuration information relevant to an open file. name specifies the configuration value to retrieve.
17	os.fstat(fd) Return status for file descriptor fd, like stat().
18	os.fstatvfs(fd) Return information about the filesystem containing the file associated with file descriptor fd, like statvfs().

19	os.fsync(fd) Force write of file with filedescriptor fd to disk.
20	os.ftruncate(fd, length) Truncate the file corresponding to file descriptor fd, so that it is at most length bytes in size.
21	os.getcwd() Return a string representing the current working directory.
22	os.getcwdu() Return a Unicode object representing the current working directory.
23	os.isatty(fd) Return True if the file descriptor fd is open and connected to a tty(-like) device, else False.
24	os.lchflags(path, flags) Set the flags of path to the numeric flags, like chflags(), but do not follow symbolic links.
25	os.lchmod(path, mode) Change the mode of path to the numeric mode.
26	os.lchown(path, uid, gid) Change the owner and group id of path to the numeric uid and gid. This function will not follow symbolic links.
27	os.link(src, dst) Create a hard link pointing to src named dst.
28	os.listdir(path) Return a list containing the names of the entries in the directory given by path.
29	os.lseek(fd, pos, how) Set the current position of file descriptor fd to position pos, modified by how.
30	os.lstat(path) Like stat(), but do not follow symbolic links.
31	os.major(device) Extract the device major number from a raw device number.
32	os.makedev(major, minor) Compose a raw device number from the major and minor device numbers.
33	os.makedirs(path[, mode]) Recursive directory creation function.
34	os.minor(device) Extract the device minor number from a raw device number .
35	os.mkdir(path[, mode]) Create a directory named path with numeric mode mode.
36	os.mkfifo(path[, mode]) Create a FIFO (a named pipe) named path with numeric mode mode. The default mode is 0666 (octal).
37	os.mknod(filename[, mode=0600, device]) Create a filesystem node (file, device special file or named pipe) named filename.
38	os.open(file, flags[, mode]) Open the file file and set various flags according to flags and possibly its mode according to mode.
39	os.openpty() Open a new pseudo-terminal pair. Return a pair of file descriptors (master, slave) for the pty and the tty, respectively.
40	os.pathconf(path, name) Return system configuration information relevant to a named file.

41	os.pipe() Create a pipe. Return a pair of file descriptors (r, w) usable for reading and writing, respectively.
42	os.popen(command[, mode[, bufsize]]) Open a pipe to or from command.
43	os.read(fd, n) Read at most n bytes from file descriptor fd. Return a string containing the bytes read. If the end of the file referred to by fd has been reached, an empty string is returned.
44	os.readlink(path) Return a string representing the path to which the symbolic link points.
45	os.remove(path) Remove the file path.
46	os.removedirs(path) Remove directories recursively.
47	os.rename(src, dst) Rename the file or directory src to dst.
48	os.renames(old, new) Recursive directory or file renaming function.
49	os.rmdir(path) Remove the directory path
50	os.stat(path) Perform a stat system call on the given path.
51	os.stat_float_times([newvalue]) Determine whether stat_result represents time stamps as float objects.
52	os.statvfs(path) Perform a statvfs system call on the given path.
53	os.symlink(src, dst) Create a symbolic link pointing to src named dst.
54	os.tcgetpgrp(fd) Return the process group associated with the terminal given by fd (an open file descriptor as returned by open()).
55	os.tcsetpgrp(fd, pg) Set the process group associated with the terminal given by fd (an open file descriptor as returned by open()) to pg.
56	os.tempnam([dir[, prefix]]) Return a unique path name that is reasonable for creating a temporary file.
57	os.tmpfile() Return a new file object opened in update mode (w+b).
58	os.tmpnam() Return a unique path name that is reasonable for creating a temporary file.
59	os.ttyname(fd) Return a string which specifies the terminal device associated with file descriptor fd. If fd is not associated with a terminal device, an exception is raised.
60	os.unlink(path) Remove the file path.
61	os.utime(path, times) Set the access and modified times of the file specified by path.

62	os.walk(top[, topdown=True[, onerror=None[, followlinks=False]]]) Generate the file names in a directory tree by walking the tree either top-down or bottom-up.
63	os.write(fd, str) Write the string str to file descriptor fd. Return the number of bytes actually written.

Here is a list all the standard Exceptions available in Python:

EXCEPTION NAME	DESCRIPTION
Exception	Base class for all exceptions
StopIteration	Raised when the next() method of an iterator does not point to any object.
SystemExit	Raised by the sys.exit() function.
StandardError	Base class for all built-in exceptions except StopIteration and SystemExit.
ArithmeticError	Base class for all errors that occur for numeric calculation.
OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
FloatingPointError	Raised when a floating point calculation fails.
ZeroDivisonError	Raised when division or modulo by zero takes place for all numeric types.
AssertionError	Raised in case of failure of the Assert statement.
AttributeError	Raised in case of failure of attribute reference or assignment.
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
ImportError	Raised when an import statement fails.
KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.
LookupError	Base class for all lookup errors.
IndexError	Raised when an index is not found in a sequence.
KeyError	Raised when the specified key is not found in the dictionary.
NameError	Raised when an identifier is not found in the local or global namespace.
UnboundLocalError	Raised when trying to access a local variable in a function or method but no value has been assigned to it.
EnvironmentError	Base class for all exceptions that occur outside the Python environment.
IOError	Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
OSError	Raised for operating systemrelated errors.
SyntaxError	Raised when there is an error in Python syntax.
IndentationError	Raised when indentation is not specified properly.

SystemError	Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
SystemExit	Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
TypeError	Raised when an operation or function is attempted that is invalid for the specified data type.
ValueError	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
RuntimeError	Raised when a generated error does not fall into any category.
NotImplementedError	Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.