**Built-in Functions with Dictionary**

Built-in functions like all(), any(), len(), cmp(), sorted() etc. are commonly used with dictionary to perform different tasks.

|  |  |
| --- | --- |
| Built-in Functions with Dictionary | |
| Function | Description |
| [all()](https://www.programiz.com/python-programming/methods/built-in/all) | Return True if all keys of the dictionary are true (or if the dictionary is empty). |
| [any()](https://www.programiz.com/python-programming/methods/built-in/any) | Return True if any key of the dictionary is true. If the dictionary is empty, return False. |
| [len()](https://www.programiz.com/python-programming/methods/built-in/len) | Return the length (the number of items) in the dictionary. |
| cmp() | Compares items of two dictionaries. |
| [sorted()](https://www.programiz.com/python-programming/methods/built-in/sorted) | Return a new sorted list of keys in the dictionary. |

Here are some examples that uses built-in functions to work with dictionary.

|  |
| --- |
| squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}  print(len(squares))  # Output: 5  print(sorted(squares))  # Output: [1, 3, 5, 7, 9] |

**Module**

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.

Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

There are different ways to import modules:

1: import module1[, module2[,... moduleN]

2: from modname import name1[, name2[, ... nameN]]

3: from modname import \*

1. You can use any Python source file as a module by executing an import statement in some other Python source file.
2. Python's *from* statement lets you import specific attributes from a module into the current namespace.
3. It is also possible to import all names from a module into the current namespace by using that import statement.

**Example**

def print\_func( par ):

print "Hello : ", par

return

--------------------------------------

#!/usr/bin/python

# Import module support

import support

# Now you can call defined function that module as follows

support.print\_func("Zara")

When the above code is executed, it produces the following result −

Hello : Zara

A module is loaded only once, regardless of the number of times it is imported. This prevents the module execution from happening over and over again if multiple imports occur.

**Packages in Python**

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and subpackages and sub-subpackages, and so on.

Example:

Having next structure, the folder Phone is a package, is important put a file \_\_init\_\_.py

Phone/Isdn.py file having function Isdn()

Phone/G3.py file having function G3()

Phone/\_\_init\_\_.py

To make all of your functions available when you've imported Phone, you need to put explicit import statements in \_\_init\_\_.py as follows −

from Pots import Pots

from Isdn import Isdn

from G3 import G3

After you add these lines to \_\_init\_\_.py, you have all of these classes available when you import the Phone package.

#!/usr/bin/python

# Now import your Phone Package.

import Phone

Phone.Pots()

Phone.Isdn()

Phone.G3()

**Regular Expressions**

As in other language python allow regular expressions.

A brief explanation of the format of regular expressions follows. For further information and a gentler presentation, consult the [Regular Expression HOWTO](https://docs.python.org/2/howto/regex.html#regex-howto).

Regular expressions can contain both special and ordinary characters. Most ordinary characters, like 'A', 'a', or '0', are the simplest regular expressions; they simply match themselves. You can concatenate ordinary characters, so last matches the string 'last'. (In the rest of this section, we’ll write RE’s in this special style, usually without quotes, and strings to be matched 'in single quotes'.)

Some characters, like '|' or '(', are special. Special characters either stand for classes of ordinary characters, or affect how the regular expressions around them are interpreted. Regular expression pattern strings may not contain null bytes, but can specify the null byte using the \number notation, e.g., '\x00'.

Repetition qualifiers (\*, +, ?, {m,n}, etc) cannot be directly nested. This avoids ambiguity with the non-greedy modifier suffix ?, and with other modifiers in other implementations. To apply a second repetition to an inner repetition, parentheses may be used. For example, the expression (?:a{6})\* matches any multiple of six 'a' characters.

The special characters are:

|  |  |
| --- | --- |
| char | description |
| '.' | (Dot.) In the default mode, this matches any character except a newline. If the [**DOTALL**](https://docs.python.org/2/library/re.html#re.DOTALL) flag has been specified, this matches any character including a newline. |
| '^' | (Caret.) Matches the start of the string, and in [**MULTILINE**](https://docs.python.org/2/library/re.html#re.MULTILINE) mode also matches immediately after each newline. |
| '$' | Matches the end of the string or just before the newline at the end of the string, and in [**MULTILINE**](https://docs.python.org/2/library/re.html#re.MULTILINE) mode also matches before a newline.foo matches both ‘foo’ and ‘foobar’, while the regular expression foo$ matches only ‘foo’. More interestingly, searching for foo.$ in 'foo1\nfoo2\n' matches ‘foo2’ normally, but ‘foo1’ in [**MULTILINE**](https://docs.python.org/2/library/re.html#re.MULTILINE) mode; searching for a single $ in 'foo\n' will find two (empty) matches: one just before the newline, and one at the end of the string. |
| '\*' | Causes the resulting RE to match 0 or more repetitions of the preceding RE, as many repetitions as are possible. ab\* will match ‘a’, ‘ab’, or ‘a’ followed by any number of ‘b’s. |
| '+' | Causes the resulting RE to match 1 or more repetitions of the preceding RE. ab+ will match ‘a’ followed by any non-zero number of ‘b’s; it will not match just ‘a’. |
| '?' | Causes the resulting RE to match 0 or 1 repetitions of the preceding RE. ab? will match either ‘a’ or ‘ab’ |
| \*?, +?, ?? | The '\*', '+', and '?' qualifiers are all *greedy*; they match as much text as possible. Sometimes this behaviour isn’t desired; if the RE<.\*> is matched against <a> b <c>, it will match the entire string, and not just <a>. Adding ? after the qualifier makes it perform the match in *non-greedy* or *minimal* fashion; as *few* characters as possible will be matched. Using the RE <.\*?> will match only <a>. |
| {m} | Specifies that exactly *m* copies of the previous RE should be matched; fewer matches cause the entire RE not to match. For example, a{6} will match exactly six 'a' characters, but not five. |
| {m,n} | Causes the resulting RE to match from *m* to *n* repetitions of the preceding RE, attempting to match as many repetitions as possible. For example, a{3,5} will match from 3 to 5 'a' characters. Omitting *m* specifies a lower bound of zero, and omitting *n* specifies an infinite upper bound. As an example, a{4,}b will match aaaab or a thousand 'a' characters followed by a b, but not aaab. The comma may not be omitted or the modifier would be confused with the previously described form. |
| {m,n}? | Causes the resulting RE to match from *m* to *n* repetitions of the preceding RE, attempting to match as *few* repetitions as possible. This is the non-greedy version of the previous qualifier. For example, on the 6-character string 'aaaaaa', a{3,5} will match 5 'a' characters, while a{3,5}? will only match 3 characters. |
| '\' | Either escapes special characters (permitting you to match characters like '\*', '?', and so forth), or signals a special sequence; special sequences are discussed below  Note: If you’re not using a raw string to express the pattern, remember that Python also uses the backslash as an escape sequence in string literals; if the escape sequence isn’t recognized by Python’s parser, the backslash and subsequent character are included in the resulting string. However, if Python would recognize the resulting sequence, the backslash should be repeated twice. This is complicated and hard to understand, so it’s highly recommended that you use raw strings for all but the simplest expressions. |
| [] | Used to indicate a set of characters. In a set:   * Characters can be listed individually, e.g. [amk] will match 'a', 'm', or 'k'. * Ranges of characters can be indicated by giving two characters and separating them by a '-', for example [a-z] will match any lowercase ASCII letter, [0-5][0-9] will match all the two-digits numbers from 00 to 59, and [0-9A-Fa-f] will match any hexadecimal digit. If - is escaped (e.g. [a\-z]) or if it’s placed as the first or last character (e.g. [a-]), it will match a literal '-'. * Special characters lose their special meaning inside sets. For example, [(+\*)] will match any of the literal characters '(', '+', '\*', or ')'. * Character classes such as \w or \S (defined below) are also accepted inside a set, although the characters they match depends on whether [**LOCALE**](https://docs.python.org/2/library/re.html#re.LOCALE) or [**UNICODE**](https://docs.python.org/2/library/re.html#re.UNICODE) mode is in force. * Characters that are not within a range can be matched by *complementing* the set. If the first character of the set is '^', all the characters that are *not* in the set will be matched. For example, [^5] will match any character except '5', and [^^] will match any character except '^'. ^ has no special meaning if it’s not the first character in the set.   To match a literal ']' inside a set, precede it with a backslash, or place it at the beginning of the set. For example, both [()[\]{}]and []()[{}] will both match a parenthesis. |
| '|' | A|B, where A and B can be arbitrary REs, creates a regular expression that will match either A or B. An arbitrary number of REs can be separated by the '|' in this way. This can be used inside groups (see below) as well. As the target string is scanned, REs separated by '|' are tried from left to right. When one pattern completely matches, that branch is accepted. This means that once A matches, B will not be tested further, even if it would produce a longer overall match. In other words, the '|' operator is never greedy. To match a literal '|', use \|, or enclose it inside a character class, as in [|] |
| (...) | Matches whatever regular expression is inside the parentheses, and indicates the start and end of a group; the contents of a group can be retrieved after a match has been performed, and can be matched later in the string with the \number special sequence, described below. To match the literals '(' or ')', use \( or \), or enclose them inside a character class: [(] [)] |
| (?...) | This is an extension notation (a '?' following a '(' is not meaningful otherwise). The first character after the '?' determines what the meaning and further syntax of the construct is. Extensions usually do not create a new group; (?P<name>...) is the only exception to this rule. Following are the currently supported extensions |
| (?iLmsux) | (One or more letters from the set 'i', 'L', 'm', 's', 'u', 'x'.) The group matches the empty string; the letters set the corresponding flags: **[re.I](https://docs.python.org/2/library/re.html" \l "re.I" \o "re.I)** (ignore case), **[re.L](https://docs.python.org/2/library/re.html" \l "re.L" \o "re.L)** (locale dependent), **[re.M](https://docs.python.org/2/library/re.html" \l "re.M" \o "re.M)** (multi-line), **[re.S](https://docs.python.org/2/library/re.html" \l "re.S" \o "re.S)** (dot matches all), **[re.U](https://docs.python.org/2/library/re.html" \l "re.U" \o "re.U)** (Unicode dependent), and **[re.X](https://docs.python.org/2/library/re.html" \l "re.X" \o "re.X)** (verbose), for the entire regular expression. (The flags are described in [Module Contents](https://docs.python.org/2/library/re.html#contents-of-module-re).) This is useful if you wish to include the flags as part of the regular expression, instead of passing a *flag* argument to the **[re.compile()](https://docs.python.org/2/library/re.html" \l "re.compile" \o "re.compile)** function.  Note that the (?x) flag changes how the expression is parsed. It should be used first in the expression string, or after one or more whitespace characters. If there are non-whitespace characters before the flag, the results are undefined |
| (?:...) | A non-capturing version of regular parentheses. Matches whatever regular expression is inside the parentheses, but the substring matched by the group *cannot* be retrieved after performing a match or referenced later in the pattern |
| (?P<name>...) | Similar to regular parentheses, but the substring matched by the group is accessible via the symbolic group name *name*. Group names must be valid Python identifiers, and each group name must be defined only once within a regular expression. A symbolic group is also a numbered group, just as if the group were not named. |

Named groups can be referenced in three contexts. If the pattern is (?P<quote>['"]).\*?(?P=quote) (i.e. matching a string quoted with either single or double quotes):

| **Context of reference to group “quote”** | **Ways to reference it** |
| --- | --- |
| in the same pattern itself | * (?P=quote) (as shown) * \1 |
| when processing match object m | * m.group('quote') * m.end('quote') (etc.) |
| in a string passed to the repl argument of re.sub() | * \g<quote> * \g<1> * \1 |

|  |  |
| --- | --- |
| (?P=name) | A backreference to a named group; it matches whatever text was matched by the earlier group named *name*. |
| (?#...) | A comment; the contents of the parentheses are simply ignored. |
| (?=...) | Matches if ... matches next, but doesn’t consume any of the string. This is called a lookahead assertion. For example, Isaac (?=Asimov)will match 'Isaac ' only if it’s followed by 'Asimov'. |
| (?!...) | Matches if ... doesn’t match next. This is a negative lookahead assertion. For example, Isaac (?!Asimov) will match 'Isaac ' only if it’s *not* followed by 'Asimov' |
| (?<=...) | Matches if the current position in the string is preceded by a match for ... that ends at the current position. This is called a *positive lookbehind assertion*. (?<=abc)def will find a match in abcdef, since the lookbehind will back up 3 characters and check if the contained pattern matches. The contained pattern must only match strings of some fixed length, meaning that abc or a|b are allowed, but a\* and a{3,4} are not. Group references are not supported even if they match strings of some fixed length. Note that patterns which start with positive lookbehind assertions will not match at the beginning of the string being searched; you will most likely want to use the [**search()**](https://docs.python.org/2/library/re.html#re.search)function rather than the [**match()**](https://docs.python.org/2/library/re.html#re.match) function. |
| (?<!...) | Matches if the current position in the string is not preceded by a match for .... This is called a negative lookbehind assertion. Similar to positive lookbehind assertions, the contained pattern must only match strings of some fixed length and shouldn’t contain group references. Patterns which start with negative lookbehind assertions may match at the beginning of the string being searched. |
| (?(id/name)yes-pattern|no-pattern) | Will try to match with yes-pattern if the group with given id or name exists, and with no-pattern if it doesn’t. no-pattern is optional and can be omitted. For example,(<)?(\w+@\w+(?:\.\w+)+)(?(1)>) is a poor email matching pattern, which will match with '<user@host.com>' as well as 'user@host.com', but not with '<user@host.com'. |

The special sequences consist of '\' and a character from the list below. If the ordinary character is not on the list, then the resulting RE will match the second character. For example, \$ matches the character '$'.

|  |  |
| --- | --- |
| \number | Matches the contents of the group of the same number. Groups are numbered starting from 1. For example, (.+) \1 matches 'the the' or '55 55', but not 'thethe' (note the space after the group). This special sequence can only be used to match one of the first 99 groups. If the first digit of number is 0, or number is 3 octal digits long, it will not be interpreted as a group match, but as the character with octal value number. Inside the '[' and ']' of a character class, all numeric escapes are treated as characters. |
| \A | Matches only at the start of the string. |
| \b | Matches the empty string, but only at the beginning or end of a word. A word is defined as a sequence of alphanumeric or underscore characters, so the end of a word is indicated by whitespace or a non-alphanumeric, non-underscore character. Note that formally, \b is defined as the boundary between a \wand a \W character (or vice versa), or between \w and the beginning/end of the string, so the precise set of characters deemed to be alphanumeric depends on the values of the UNICODE and LOCALE flags. For example, r'\bfoo\b' matches 'foo', 'foo.', '(foo)', 'bar foo baz'but not 'foobar' or 'foo3'. Inside a character range, \brepresents the backspace character, for compatibility with Python’s string literals. |
| \B | Matches the empty string, but only when it is not at the beginning or end of a word. This means that r'py\B'matches 'python', 'py3', 'py2', but not 'py', 'py.', or 'py!'. \B is just the opposite of \b, so is also subject to the settings of LOCALE and UNICODE. |
| \d | When the [**UNICODE**](https://docs.python.org/2/library/re.html#re.UNICODE) flag is not specified, matches any decimal digit; this is equivalent to the set [0-9]. With [**UNICODE**](https://docs.python.org/2/library/re.html#re.UNICODE), it will match whatever is classified as a decimal digit in the Unicode character properties database. |
| \D | When the [**UNICODE**](https://docs.python.org/2/library/re.html#re.UNICODE) flag is not specified, matches any non-digit character; this is equivalent to the set [^0-9]. With [**UNICODE**](https://docs.python.org/2/library/re.html#re.UNICODE), it will match anything other than character marked as digits in the Unicode character properties database. |
| \s | When the [**UNICODE**](https://docs.python.org/2/library/re.html#re.UNICODE) flag is not specified, it matches any whitespace character, this is equivalent to the set [\t\n\r\f\v]. The [**LOCALE**](https://docs.python.org/2/library/re.html#re.LOCALE) flag has no extra effect on matching of the space. If [**UNICODE**](https://docs.python.org/2/library/re.html#re.UNICODE) is set, this will match the characters [ \t\n\r\f\v] plus whatever is classified as space in the Unicode character properties database. |
| \S | When the [**UNICODE**](https://docs.python.org/2/library/re.html#re.UNICODE) flag is not specified, matches any non-whitespace character; this is equivalent to the set [^ \t\n\r\f\v] The [**LOCALE**](https://docs.python.org/2/library/re.html#re.LOCALE) flag has no extra effect on non-whitespace match. If [**UNICODE**](https://docs.python.org/2/library/re.html#re.UNICODE) is set, then any character not marked as space in the Unicode character properties database is matched. |
| \w | When the [**LOCALE**](https://docs.python.org/2/library/re.html#re.LOCALE) and [**UNICODE**](https://docs.python.org/2/library/re.html#re.UNICODE) flags are not specified, matches any alphanumeric character and the underscore; this is equivalent to the set [a-zA-Z0-9\_]. With [**LOCALE**](https://docs.python.org/2/library/re.html#re.LOCALE), it will match the set [0-9\_] plus whatever characters are defined as alphanumeric for the current locale. If [**UNICODE**](https://docs.python.org/2/library/re.html#re.UNICODE) is set, this will match the characters [0-9\_] plus whatever is classified as alphanumeric in the Unicode character properties database. |
| \W | When the [**LOCALE**](https://docs.python.org/2/library/re.html#re.LOCALE) and [**UNICODE**](https://docs.python.org/2/library/re.html#re.UNICODE) flags are not specified, matches any non-alphanumeric character; this is equivalent to the set [^a-zA-Z0-9\_]. With [**LOCALE**](https://docs.python.org/2/library/re.html#re.LOCALE), it will match any character not in the set [0-9\_], and not defined as alphanumeric for the current locale. If [**UNICODE**](https://docs.python.org/2/library/re.html#re.UNICODE) is set, this will match anything other than [0-9\_]plus characters classified as not alphanumeric in the Unicode character properties database. |
| \Z | Matches only at the end of the string. |