**Modules and conditional statement**

**Modules**

The [Python interactive shell](https://www.tutorialsteacher.com/python/python-interective-shell) has a number of built-in functions. They are loaded automatically as a shell starts and are always available, and we load them by importing them the use the dir() function to check all the function under it, use the link below to check all module in python

# <https://www.tutorialsteacher.com/python/python-builtin-modules> Python - Built-in Modules

The [Python interactive shell](https://www.tutorialsteacher.com/python/python-interective-shell) has a number of built-in functions. They are loaded automatically as a shell starts and are always available, such as [print()](https://www.tutorialsteacher.com/python/print-method) and [input()](https://www.tutorialsteacher.com/python/input-function) for I/O, number conversion functions [int()](https://www.tutorialsteacher.com/python/int-method), [float()](https://www.tutorialsteacher.com/python/float-method), [complex()](https://www.tutorialsteacher.com/python/complex-method), data type conversions [list()](https://www.tutorialsteacher.com/python/list-method), [tuple()](https://www.tutorialsteacher.com/python/tuple-method), [set()](https://www.tutorialsteacher.com/python/set-method), etc.

In addition to built-in functions, a large number of pre-defined functions are also available as a part of libraries bundled with Python distributions. These functions are defined in [modules](https://www.tutorialsteacher.com/python/python-module) are called built-in modules.

Built-in modules are written in C and integrated with the Python shell. Each built-in module contains resources for certain system-specific functionalities such as OS management, disk IO, etc. The standard library also contains many Python scripts (with the .py extension) containing useful utilities.

To display a list of all available modules, use the following command in the Python console:

Python IDLE

>>> help('modules')

IPython \_weakrefset heapq secrets

\_\_future\_\_ \_winapi hmac select

\_abc abc html selectors

\_ast aifc http setuptools

\_asyncio antigravity idlelib shelve

\_bisect argparse imaplib shlex

\_blake2 array imghdr shutil

\_bootlocale ast imp signal

\_bz2 asynchat importlib simplegeneric

\_codecs asyncio ind site

\_codecs\_cn asyncore inspect six

\_codecs\_hk atexit io smtpd

\_codecs\_iso2022 audioop ipaddress smtplib

\_codecs\_jp autoreload ipython\_genutils sndhdr

\_codecs\_kr backcall itertools socket

\_codecs\_tw base64 jedi socketserver

\_collections bdb json sqlite3

\_collections\_abc binascii keyword sre\_compile

\_compat\_pickle binhex lib2to3 sre\_constants

\_compression bisect linecache sre\_parse

\_contextvars builtins locale ssl

\_csv bz2 logging stat

\_ctypes cProfile lzma statistics

\_ctypes\_test calendar macpath storemagic

\_datetime cgi mailbox string

\_decimal cgitb mailcap stringprep

\_distutils\_findvs chunk marshal struct

\_dummy\_thread cmath math subprocess

\_elementtree cmd mimetypes sunau

\_functools code mmap symbol

\_hashlib codecs modulefinder sympyprinting

\_heapq codeop msilib symtable

\_imp collections msvcrt sys

\_io colorama multiprocessing sysconfig

\_json colorsys netrc tabnanny

\_locale compileall nntplib tarfile

\_lsprof concurrent nt telnetlib

\_lzma configparser ntpath tempfile

\_markupbase contextlib nturl2path test

\_md5 contextvars numbers tests

\_msi copy opcode textwrap

\_multibytecodec copyreg operator this

\_multiprocessing crypt optparse threading

\_opcode csv os time

\_operator ctypes parser timeit

\_osx\_support curses parso tkinter

\_overlapped cythonmagic pathlib token

\_pickle dataclasses pdb tokenize

\_py\_abc datetime pickle trace

\_pydecimal dbm pickleshare traceback

\_pyio decimal pickletools tracemalloc

\_queue decorator pip traitlets

\_random difflib pipes tty

\_sha1 dis pkg\_resources turtle

\_sha256 distutils pkgutil turtledemo

\_sha3 doctest platform types

\_sha512 dummy\_threading plistlib typing

\_signal easy\_install poplib unicodedata

\_sitebuiltins email posixpath unittest

\_socket encodings pprint urllib

\_sqlite3 ensurepip profile uu

\_sre enum prompt\_toolkit uuid

\_ssl errno pstats venv

\_stat faulthandler pty warnings

\_string filecmp py\_compile wave

\_strptime fileinput pyclbr wcwidth

\_struct fnmatch pydoc weakref

\_symtable formatter pydoc\_data webbrowser

\_testbuffer fractions pyexpat winreg

\_testcapi ftplib pygments winsound

\_testconsole functools queue wsgiref

\_testimportmultiple gc quopri xdrlib

\_testmultiphase genericpath random xml

\_thread getopt re xmlrpc

\_threading\_local getpass reprlib xxsubtype

\_tkinter gettext rlcompleter zipapp

\_tracemalloc glob rmagic zipfile

\_warnings gzip runpy zipimport

\_weakref hashlib sched zlib

Enter any module name to get more help. Or, type "modules spam" to search

for modules whose name or summary contain the string "spam".

**Conditional Statement**

Every day we are faced with decisions

1. Should I drive or take the bus?
2. Should I cook at home or go out for dinner?
3. Which laptop should I buy?

we will be looking at if, if-else and if-elif-else

**If statements**

If statements  allow you to specify code that only executes if a specific condition is true

**If-else  statements**

If statements  allow you to specify code that only executes if a specific condition is true while The code in the***else***statement is only executed if the condition is NOT true

**If-elif-else  statements**

Sometimes there are multiple conditions that affect the outcome of a decision

â€¢If you are in England say hello, if you are in Germany say guten tag, if you are in France say bonjour, â€¦

â€¢If you win the lottery and the prize is over a million dollars then retire to a life of luxury

â€¢If it is Monday, check to see if there is fresh coffee. If there is no fresh coffee go to the nearest cafÃ©

The â€œelifâ€ allows you to check for different values  
Read more with this link <https://www.tutorialsteacher.com/python/python-if-elif>

# Python - if, elif, else Conditions

By default, statements in the script are executed sequentially from the first to the last. If the processing logic requires so, the sequential flow can be altered in two ways:

Python uses the if keyword to implement decision control. Python's syntax for executing a block conditionally is as below:

Syntax:

if [boolean expression]:

statement1

statement2

...

statementN

Any Boolean expression evaluating to True or False appears after the if keyword. Use the : symbol and press Enter after the expression to start a block with an increased indent. One or more statements written with the same level of indent will be executed if the Boolean expression evaluates to True.

To end the block, decrease the indentation. Subsequent statements after the block will be executed out of the if condition. The following example demonstrates the if condition.

Example: if Condition

price = 50

if price < 100:

print("price is less than 100")

Output

price is less than 100

In the above example, the expression price < 100 evaluates to True, so it will execute the block. The if block starts from the new line after : and all the statements under the if condition starts with an increased indentation, either space or tab. Above, the if block contains only one statement. The following example has multiple statements in the if condition.

Example: Multiple Statements in the if Block

price = 50

quantity = 5

if price\*quantity < 500:

print("price\*quantity is less than 500")

print("price = ", price)

print("quantity = ", quantity)

Output

price\*quantity is less than 500

price = 50

quantity = 5

Above, the if condition contains multiple statements with the same indentation. If all the statements are not in the same indentation, either space or a tab then it will raise an IdentationError.

Example: Invalid Indentation in the Block

price = 50

quantity = 5

if price\*quantity < 500:

print("price is less than 500")

print("price = ", price)

print("quantity = ", quantity)

Output

print("quantity = ", quantity)

^

IdentationError: unexpected indent

The statements with the same indentation level as if condition will not consider in the if block. They will consider out of the if condition.

Example: Out of Block Statements

price = 50

quantity = 5

if price\*quantity < 100:

print("price is less than 500")

print("price = ", price)

print("quantity = ", quantity)

print("No if block executed.")

Output

No if block executed.

The following example demonstrates multiple if conditions.

Example: Multiple if Conditions

price = 100

if price > 100:

print("price is greater than 100")

if price == 100:

print("price is 100")

if price < 100:

print("price is less than 100")

Output

price is 100

Notice that each if block contains a statement in a different indentation, and that's valid because they are different from each other.

Note

It is recommended to use 4 spaces or a tab as the default indentation level for more readability.