# Basic Data Types

Try the following in the interactive interpreter

**Integers and Floats**

>>> 5

>>> type(5)

>>> type(5.0)

>>> 5 + 3

>>> 5 \* 2

>>> 5 \*\* 2

>>> 5 / 2

>>> 5 // 2

**Strings**

>>> "a"

>>> type("a")

>>> "a" + "b"

>>> "a" \* 3

>>> "a" \* "b"

>>> "1" + 2

>>> a = "hi there!"

>>> a[0]

>>> a[100]

>>> a[-1]

>>> a[2:5]

**Lists**

>>> [1, 2]

>>> type([1, 2])

>>> [1, 2] + [3, 4]

>>> [1] \* 4

>>> [1, "abc"]

>>> a = [1, 2, 3, 4, 5]

>>> a[0]

>>> a[2]

>>> a[-1]

>>> a[1:3]

>>> a[1:-2]

>>> a[:2]

>>> a[2:]

>>> a[:-1]

>>> a[:]

>>> a = [[1, 2], [3, 4]]

>>> a[0]

>>> a[0][1]

>>> a = [1, 2, 3, 4, 5]

>>> a[1] = 7

>>> print(a)

>>> a[100]

**Dictionaries**

>>> a = {"name": "python", "id": 231}

>>> type(a)

>>> a["name"]

>>> a["id"]

>>> a = {"name": "python", 42: "answer"}

>>> a["name"]

>>> a[42]

>>> a = {"name": "python", "dates": ["10 Aug", "11 Aug"]}

>>> a["dates"]

>>> a["dates"][-1]

>>> a["name"] = "python training"

>>> print(a)

>>> a["type"]

>>> a["type"] = "development"

>>> a["type"]

**Tuples**

>>> point = (1, 2)

>>> type(point)

>>> point[1]

>>> point[2]

>>> training = ("Python Training", ["10 Aug", "11 Aug"])

>>> training[1][1]

**Basic String Methods**

>>> a = "this is a string"

>>> dir("a")

>>> len(a)

>>> a.split()

>>> "-".join(["a", "b", "c"])

>>> a.upper()

>>> a.title()

>>> a.startswith("this")

>>> " a string ".strip()

>>> a.index("a")

>>> a.index("z")

>>> a.replace(" ", ".")

**Basic List Methods**

>>> a = [1, 2, 3]

>>> dir(a)

>>> len(a)

>>> a.append(4)

>>> a.pop()

>>> a.index(3)

>>> a.index(100)

>>> a.remove(2)

>>> print(a)

>>> a.remove(100)

>>> a.insert(1, 15)

>>> print(a)

**Basic Dictionary Methods**

>>> a = {"name": "Sid", "marks": 10}

>>> dir(a)

>>> list(a.items())

>>> list(a.keys())

>>> list(a.values())

**Basic List Sorting**

>>> a = [("Sid", 10), ("Priya", 15), ("Akshay", 5)]

>>> a.sort()

>>> print(a)

>>> def marks\_field(person):

... return person[1]

...

>>> a.sort(key=marks\_field, reverse=True)

>>> print(a)

# Date and Time

from datetime import datetime

jan\_5\_noon = datetime(2015, 1, 5, 12, 0, 0)

jan\_5\_noon.[day/date/month/year/hour/minute/second/microsecond]

jan\_5\_noon.strftime("%d/%m/%Y") -> String output based on format string

**datetime.now()** -> Get current datetime

**datetime.fromtimestamp()** -> Convert a long timestamp into datetime object

**datetime.strptime("2015-01-11 12:00:00", "%Y-%m-%d %H:%M:%s")** -> Convert string to datetime object

from datetime import timedelta

jan\_5\_noon + timedelta(1) -> jan 6 noon

jan\_5\_noon + timedelta(1/24) -> jan 5 1:00pm

td = datetime.now() - jan\_5\_noon -> timedelta between the dates

td.days -> Number of days in timedelta

**Date String Format Syntax**

| Directive | Meaning |
| --- | --- |
| %a | Weekday as locale’s abbreviated name. |
| %A | Weekday as locale’s full name. |
| %w | Weekday as a decimal number, where 0 is Sunday and 6 is Saturday. |
| %d | Day of the month as a zero-padded decimal number. |
| %b | Month as locale’s abbreviated name. |
| %B | Month as locale’s full name. |
| %m | Month as a zero-padded decimal number. |
| %y | Year without century as a zero-padded decimal number. |
| %Y | Year with century as a decimal number. |
| %H | Hour (24-hour clock) as a zero-padded decimal number. |
| %I | Hour (12-hour clock) as a zero-padded decimal number. |
| %p | Locale’s equivalent of either AM or PM. |
| %M | Minute as a zero-padded decimal number. |
| %S | Second as a zero-padded decimal number. |
| %f | Microsecond as a decimal number, zero-padded on the left. |
| %z | UTC offset in the form +HHMM or -HHMM (empty string if the the object is naive). |
| %Z | Time zone name (empty string if the object is naive). |
| %j | Day of the year as a zero-padded decimal number. |
| %U | Week number of the year (Sunday as the first day of the week) |
| %W | Week number of the year (Monday as the first day of the week) |
| %c | Locale’s appropriate date and time representation. |
| %x | Locale’s appropriate date representation. |
| %X | Locale’s appropriate time representation. |
| %% | A literal '%' character. |

# Built-in Functions

|  |  | Built-in Functions |  |  |
| --- | --- | --- | --- | --- |
| [abs()](https://docs.python.org/3/library/functions.html#abs) | [dict()](https://docs.python.org/3/library/functions.html#func-dict) | [help()](https://docs.python.org/3/library/functions.html#help) | [min()](https://docs.python.org/3/library/functions.html#min) | [setattr()](https://docs.python.org/3/library/functions.html#setattr) |
| [all()](https://docs.python.org/3/library/functions.html#all) | [dir()](https://docs.python.org/3/library/functions.html#dir) | [hex()](https://docs.python.org/3/library/functions.html#hex) | [next()](https://docs.python.org/3/library/functions.html#next) | [slice()](https://docs.python.org/3/library/functions.html#slice) |
| [any()](https://docs.python.org/3/library/functions.html#any) | [divmod()](https://docs.python.org/3/library/functions.html#divmod) | [id()](https://docs.python.org/3/library/functions.html#id) | [object()](https://docs.python.org/3/library/functions.html#object) | [sorted()](https://docs.python.org/3/library/functions.html#sorted) |
| [ascii()](https://docs.python.org/3/library/functions.html#ascii) | [enumerate()](https://docs.python.org/3/library/functions.html#enumerate) | [input()](https://docs.python.org/3/library/functions.html#input) | [oct()](https://docs.python.org/3/library/functions.html#oct) | [staticmethod()](https://docs.python.org/3/library/functions.html#staticmethod) |
| [bin()](https://docs.python.org/3/library/functions.html#bin) | [eval()](https://docs.python.org/3/library/functions.html#eval) | [int()](https://docs.python.org/3/library/functions.html#int) | [open()](https://docs.python.org/3/library/functions.html#open) | [str()](https://docs.python.org/3/library/functions.html#func-str) |
| [bool()](https://docs.python.org/3/library/functions.html#bool) | [exec()](https://docs.python.org/3/library/functions.html#exec) | [isinstance()](https://docs.python.org/3/library/functions.html#isinstance) | [ord()](https://docs.python.org/3/library/functions.html#ord) | [sum()](https://docs.python.org/3/library/functions.html#sum) |
| [bytearray()](https://docs.python.org/3/library/functions.html#bytearray) | [filter()](https://docs.python.org/3/library/functions.html#filter) | [issubclass()](https://docs.python.org/3/library/functions.html#issubclass) | [pow()](https://docs.python.org/3/library/functions.html#pow) | [super()](https://docs.python.org/3/library/functions.html#super) |
| [bytes()](https://docs.python.org/3/library/functions.html#bytes) | [float()](https://docs.python.org/3/library/functions.html#float) | [iter()](https://docs.python.org/3/library/functions.html#iter) | [print()](https://docs.python.org/3/library/functions.html#print) | [tuple()](https://docs.python.org/3/library/functions.html#func-tuple) |
| [callable()](https://docs.python.org/3/library/functions.html#callable) | [format()](https://docs.python.org/3/library/functions.html#format) | [len()](https://docs.python.org/3/library/functions.html#len) | [property()](https://docs.python.org/3/library/functions.html#property) | [type()](https://docs.python.org/3/library/functions.html#type) |
| [chr()](https://docs.python.org/3/library/functions.html#chr) | [frozenset()](https://docs.python.org/3/library/functions.html#func-frozenset) | [list()](https://docs.python.org/3/library/functions.html#func-list) | [range()](https://docs.python.org/3/library/functions.html#func-range) | [vars()](https://docs.python.org/3/library/functions.html#vars) |
| [classmethod()](https://docs.python.org/3/library/functions.html#classmethod) | [getattr()](https://docs.python.org/3/library/functions.html#getattr) | [locals()](https://docs.python.org/3/library/functions.html#locals) | [repr()](https://docs.python.org/3/library/functions.html#repr) | [zip()](https://docs.python.org/3/library/functions.html#zip) |
| [compile()](https://docs.python.org/3/library/functions.html#compile) | [globals()](https://docs.python.org/3/library/functions.html#globals) | [map()](https://docs.python.org/3/library/functions.html#map) | [reversed()](https://docs.python.org/3/library/functions.html#reversed) | [\_\_import\_\_()](https://docs.python.org/3/library/functions.html#__import__) |
| [complex()](https://docs.python.org/3/library/functions.html#complex) | [hasattr()](https://docs.python.org/3/library/functions.html#hasattr) | [max()](https://docs.python.org/3/library/functions.html#max) | [round()](https://docs.python.org/3/library/functions.html#round) |  |
| [delattr()](https://docs.python.org/3/library/functions.html#delattr) | [hash()](https://docs.python.org/3/library/functions.html#hash) | [memoryview()](https://docs.python.org/3/library/functions.html#func-memoryview) | [set()](https://docs.python.org/3/library/functions.html#func-set) |  |

**type, id** – Return the type or id of an object

**print** – Print to an output stream (default console)

**input** – Read from an input stream (default console)

**int, float, bool, set, list, dict, tuple, complex, bytes, bytearray, object** – Constructors for common built in data types

**locals, globals** – Return a dict containing all the local / global variables

**\_\_import\_\_** – Import a module from a string representation

**enumerate** – Iterate returning the index as well as the object

**range** – Return a list of numbers. Mostly used to iterate by index

**iter** – Return an iterator for an object

**zip** – Iterate simultaneously through multiple iterators

**all, any** – Return True if all / any items in an iterator are True

**map, filter** – Map / Filter functionality

**sum** – Add all elements in an iterator

**super** – Call a superclass method

**isinstance, issubclass** – Returns True if one object is an instance / subclass of another

**callable** – Return True if the object can be called

**classmethod, staticmethod, property** – Decorators to define class methods, static methods and properties on object methods

**str, len, hash** – Return the string / length / hash of an object

**hasattr** – Return True if an object has an attribute

**getattr** – Get an attribute of an object

**setattr** – Set a value for an object attribute

# Boolean Operators

**Check for equality (by value!)**

==

!=

**Check for equality (by object identity)**

is

not is

**Comparison**

>

<

>=

<=

**Boolean Operations**

not

and

or

**Contains**

in

not in

# Functions

**Creating a function**

def add(num1, num2):

return num1 + num2

add(1, 5)

add("a", "b")

add(1, "a")

**Calling with keyword parameters**

add(num2=3, num1=17)

add(1, num2=12)

add(1, num1=7)

**Argument expansion**

params = (1, 2)

add(\*params)

params = {"num1": 1, "num2": 2}

add(\*\*params)

**Variable arguments**

def add(\*args):

return sum(args)

add(1, 2)

add(1, 2, 3)

def add(\*\*kwargs):

sum = 0

for value in kwargs.values():

sum = sum + value

return sum

add(num1=5, num2=7, num3=21)

**Returning multiple values**

def get\_field\_from\_db():

return True, {"name": "Sid"}

is\_success, value = get\_field\_from\_db()

print(is\_success, value)

**Default arguments values**

def add(num1=10, num2=-1):

return num1 + num2

add()

add(3)

**Lambda**

add = lambda num1, num2: num1 + num2

add(1, 3)

# Exception Handling

**Simple Exception Handling**

try:

1/0

except ZeroDivisionError:

print("You can't divide by zero")

**Handle multiple Exceptions with finally to clean up**

try:

f = open("file.txt")

data = f.read()

print(data)

except (FileNotFoundError, IOError):

print("Unable to open file")

except Exception as e:

print("Some other exception: {0}".format(str(e)))

finally:

print("Cleaning up")

if f is not None:

f.close()

**Raise an exception**

value, length = 5, 2

if value > length:

raise IndexError("Value is out of array bounds")

**Raise a custom exception**

class MyException(Exception):

pass

denominator = 0

if denominator == 0:

raise MyException("Denominator is zero")

**Re-raise an exception**

try:

1/0

except ZeroDivisionError as e:

print("You can't divide by zero")

raise e

**Exception chaining**

class MyException(Exception):

pass

try:

1/0

except ZeroDivisionError as e:

print("You can't divide by zero")

raise MyException(e)

# Files

**Read a file**

f = open("file.txt")

data = f.read()

f.close()

**Write to a file**

f = open("file.txt", "w")

f.write(data)

f.close()

**Read a binary file**

f = open("file.txt", "rb")

data = f.read()

f.close()

**Iterate over the lines of the file**

f = open("file.txt")

for line in f:

do\_something(line)

f.close()

**Ensure the file gets closed**

try:

f = open("file.txt")

do\_stuff()

finally:

if f is not None:

f.close()

**Context manager syntax**

with open("file.txt") as f:

do\_something(f)

# System and OS

import sys

dir(sys)

**sys.argv** -> List of command line parameters

**sys.exit** -> Exit to the operating system

**sys.version** -> Python version

**sys.exc\_info** -> Get stack trace & other information about an exception

import os

dir(os)

**os.system** -> Execute an operating system shell command

**os.listdir** -> Get files in a directory

**os.walk** -> Iterate through files in a directory, including sub-directories

**os.getcwd** -> Get current working directory

**os.makedirs** -> Make a directory, including intermediate directories

**os.rmdir** -> Remove a directory

**os.rename** -> Rename a file

**os.unlink** -> Delete a file

dir(os.path)

**os.path.split** -> Splits a file path into components

**os.path.isdir** -> True if path is a directory

**os.path.isfile** -> True if path is a file

**os.path.exists** -> Path exists on the filesystem

**os.path.join** -> Join path components into a full path

# Classes and Objects

**Define a class and instantiate it**

class A:

pass

obj = A()

**Class initialiser**

class A:

def \_\_init\_\_(self, name):

self.name = name

obj = A("Sid")

print(obj.name)

**Inheritance**

class B(A):

pass

**Multiple inheritance**

class C(A, B):

pass

**Methods**

class A:

def hi(self, name):

print("hello ", name)

**Instance fields**

class A:

def \_\_init\_\_(self, name):

self.name = name

def hi(self):

print("hello ", self.name)

**Class fields**

class A:

greeting = "hello!"

def hi(self):

print(greeting)

**Call superclass method**

class A:

def hi(self):

print("hello!")

class B(A):

def hi(self):

super().hi()

# Special class methods

\_\_init\_\_ -> Initialise a newly constructed class

\_\_str\_\_ -> String representation of class. Called from str() function

\_\_len\_\_ -> Length of class. Called from len() function

\_\_del\_\_ -> Called when the object is garbage collected

\_\_eq\_\_ -> Override = operator

\_\_ne\_\_ -> Override != operator

\_\_gt\_\_ -> Override > operator

\_\_ge\_\_ -> Override >= operator

\_\_lt\_\_ -> Override < operator

\_\_le\_\_ -> Override <= operator

\_\_contains\_\_ -> Override 'in' operator

\_\_getitem\_\_ -> Override [] operator. Eg: obj[key]

\_\_setitem\_\_ -> Override [] = operator. Eg: obj[key] = value

\_\_delitem\_\_ -> Override del [] operator. Eg: del obj[key]

\_\_iter\_\_ -> Return an iterator for the object

\_\_next\_\_ -> Return the next element in the iteration

\_\_enter\_\_ -> Context manager entry method

\_\_exit\_\_ -> Context manager exit method

\_\_getattr\_\_ -> Called when a missing attribute is accessed on an object

\_\_setattr\_\_ -> Called when a missing attribute is set on an object

\_\_getattribute\_\_ -> Called when any attribute is accessed on an object

\_\_setattribute\_\_ -> Called when any attribute is set on an object

# List Comprehensions

**a\_list = [x for x in some\_list]**

a\_list = []

for x in some\_list:

a\_list.append(x)

**a\_list = [func(x) for x in some\_list]**

a\_list = []

for x in some\_list:

a\_list.append(func(x))

**a\_list = [func(x) for x in some\_list if condition]**

a\_list = []

for x in some\_list:

if condition:

a\_list.append(func(x))

**a\_list = [func(x, y) for x in some\_list for y in another\_list]**

a\_list = []

for y in another\_list:

for x in some\_list:

a\_list.append(func(x, y))

**a\_list = [[func(x, y) for x in some\_list] for y in another\_list]**

a\_list = []

for y in another\_list:

temp\_list = []

for x in some\_list:

temp\_list.append(func(x, y))

a\_list.append(temp\_list)

The above list comprehension allows quickly creating 2-D nested lists