Useful Functions and Methods

Built-in:

format(numeric_value, format_specifier)

Formats a numeric value into a string according to the format specifier, which is a string that contains special characters specifying how the numeric value should be formatted. Examples of various formatting characters are “f=floating-point, e=scientific notation, %=percentage, d=integer”. A number before the formatting character will specify the field width. A number after the character “.” will format the number of decimals.

f-string

Syntax: f'…..{expression}…' where expression can be variable or any expression. Can also use formatting characters such as : d=integer, f=floating-point, e=scientific notation, %=percentage, s=string. A number before the formatting character will specify field width. A number after the character “.” will format the number of decimals. E.g. print PI with a field width of 5 with two decimals will be: `print(f'{math.pi:5.2f}')`

%

Remainder (modulo operator): Divides one number by another and gives the remainder.

//

Floor/integer division: Returns the integral part of the quotient.

len(s)

Return the length (the number of items) of a string, tuple, list, dictionary or other data structure.

int(x)

Convert a string or number to a plain integer.

float(x)

Convert a string or a number to floating point number.

str([object])

Return a string containing a nicely printable representation of an object.

range(stop)

Returns a list with integers from 0, up to but not including stop. range(3) = [0, 1, 2]. Often used in combination with for loops: for i in range(10)

range([start], stop[, step])

start: Starting number of the sequence.
stop: Generate numbers up to, but not including, this number.
step: Difference between each number in the sequence.

chr(i)

Return a string of one character whose ASCII code is the integer i. For example, chr(97) returns the string 'a'. This is the inverse of ord()

ord()

Given a string of length one, return an integer representing the Unicode code point of the character when the argument is a unicode object, or the value of the byte when the argument is an 8-bit string. For example, ord('a') returns the integer 97.

tuple(iterable)

Accepts something iterable (list, range etc.) and returns the corresponding tuple. A tuple is immutable.

if x in iterable:

Returns True if x is an item in iterable.

for idx, x in enumerate(iterable)

Enters a loop with x being one and one item from iterable. idx is an integer containing the loop number.

Exceptions:

```
try:
        # Code to test
except:
        # If code fails. E.g exception types: IOError, ValueError, ZeroDivisionError.
        # Variant: except Exception as exc # Let's you print the exception.
else:
        # Runs if no exception occurs
finally:
        # Runs regardless of prior code having succeeded or failed.
```

<u>String methods:</u>

s.isalnum()
: Returns true if the string contains only alphabetic letters or digits and is at least one character of length. Returns false otherwise.

s.isalpha()
: Returns true if the string contains only alphabetic letters, and is at least one character in length. Returns false otherwise.

s.isdigit()
: Returns true if the string contains only numeric digits and is at least one character in length. Returns false otherwise.

s.center(width)
: Return the string center justified in a string of length width.

s.ljust(width)
: Return the string left justified in a string of length width.

s.rjust(width)
: Return the string right justified in a string of length width.

s.lower()
: Returns a copy of the string with all alphabetic letters converted to lowercase.

s.upper()
: Returns a copy of the string with all alphabetic letters converted to uppercase.

s.strip()
: Returns a copy of the string with all leading and trailing white space characters removed.

s.strip(char)
: Returns a copy of the string with all instances of *char* that appear at the beginning and the end of the string removed.

s.split(str)
: Returns a list of all the words in the string, using str as the separator (splits on all whitespace if left unspecified).

s.splitlines([keepends])
: Return a list of the lines in the string, breaking at line boundaries. This method uses the universal newlines approach to splitting lines. Line breaks are not included in the resulting list unless keepends is given and true.
: Python recognizes "\r", "\n", and "\r\n" as line boundaries for 8-bit strings.

s.endswith(substring)
: The substring argument is a string. The method returns true if the string ends with substring.

s.startswith(substring)
: The substring argument is a string. The method returns true if the string starts with substring.

s.find(substring)
: The substring argument is a string. The method returns the lowest index in the string where substring is found. If substring is not found the method returns -1.

s.replace(old, new)
: The old and new arguments are both strings. The method returns a copy of the string with all instances of old replaced by new.

str.format(*args, **kwargs)
: Perform a string formatting operation. The string on which this method is called can contain literal text or replacement fields delimited by braces {}. Each replacement field contains either the numeric index of a positional argument, or the name of a keyword argument. Returns a copy of the string where each replacement field is replaced with the string value of the corresponding argument.

<u>Random methods:</u>

random.random()
: Return the next random floating-point number in the range [0.0, 1.0).

random.randint(a,b)
: Return a random integer *N* such that a <= N <= b.

random.choice(seq)
: Return a random element from the non-empty sequence seq. If seq is empty, raises IndexError.

random.randrange(start, stop [, step])
: Return a randomly selected element from range(start, stop, step).

random.uniform(a, b)
    Return a random floating-point number N such that a <= N <= b for a <= b and b <= N <= a for b < a.

List operations:

s[i:j:k]
    Return slice starting at position i extending to position j in k steps. Can also be used for strings.
item in s
    Determine whether a specified item is contained in a list.
min(list)
    Returns the item that has the lowest value in the sequence.
max(list)
    Returns the item that has the highest value in the sequence.
s.append(x)
    Append new element x to end of s. Works in_place. Returns None
s.insert(index,item)
    Insert an item into a list at a specified position given by an index.
s.index(*item*)
    Return the index of the first element in the list containing the specified item.
s.pop()
    Return last element and remove it from the list.
s.pop(i)
    Return element i and remove it from the list.
s.remove(item)
    Removes the first element containing the item. Works in_place. Returns None
s.reverse()
    Reverses the order of the items in a list. Works in_place. Returns None
s.sort()
    Rearranges the elements of a list so they appear in ascending order. Works in_place. Returns None

Sets operations:

len(s)
    Number of elements in set *s*
s.issubset(t)
    Test whether every element in *s* is in *t*
s.issuperset(t)
    Test whether every element in *t* is in *s*
s.union(t)
    New set with elements from both *s* and *t*
s.intersection(t)
    New set with elements common to *s* and *t*
s.difference(t)
    New set with elements in *s* but not in *t*
s.symmetric_difference(t)
    New set with elements in either *s* or *t* but not both
s.copy()
    New set with a shallow copy of *s*
s.update(t)
    Return set *s* with elements added from *t*
s.add(x)
    Add element *x* to set *s.* Works in_place. Returns None
s.remove(x)
    Remove x from set s; raises *KeyError* if not present. Works in_place. Returns None
s.clear()
    Remove all elements from set *s.* Works in_place. Returns None

Dictionary operations:

d.clear()
    Clears the contents of a dictionary

d.get(key, default)
: Gets the value associated with a specific key. If the key is not found, the method does not raise an exception. Instead, it returns a default value.

d.items()
: Returns all the keys in a dictionary and their associated values as a sequence of tuples.

d.keys()
: Returns all the keys in a dictionary as a sequence of tuples.

d.pop(key, default)
: Returns the value associated with a specific key and removes that key-value pair from the dictionary. If the key is not found, the method returns a default value.

d.popitem()
: Returns a randomly selected key-value pair as a tuple from the dictionary and removes that key-value pair from the dictionary.

d.values()
: Returns all the values in dictionary as a sequence of tuples.

del d[k]
: Deletes element k in d.

d.copy()
: Makes a copy of d.

Files:

open()
: Returns a file object, and is most commonly used with two arguments: open(filename, mode). Mode can be 'r' (read only), 'w' (writing only), 'a' (appending), 'r+' (both reading and writing).

f.read(size)
: Reads data from file and returns it as a string. Size is an optional and if left out the whole file will be read.

f.readline()
: Reads a single line from the file (reads until newline character (\n) is found), and returns it as a string.

f.readlines()
: Reads data from the file and returns it as a list of strings.

f.write(string)
: Writes the contents of string to file.

f.writelines(list)
: Writes the contents of a list to file

f.seek(offset, from_what)
: Move the file pointer in the file and return the new position of the file pointer. The parameter from_what can have three values: 0 – beginning of file, 1 – current position in file, and 2 – end of file. The offset parameter defines how much you will move from the from_what position.

f.tell()
: Return the position of the file pointer.

f.close()
: Close the file and free up any system resources taken up by the open file.

Pickle Library:

pickle.dump(obj,file)
: Write a pickled (serialized) representation of an obj to the open file object file.

pickle.load(file_object)
: Read a string from the open file object file and interpret it as a pickle data stream, reconstructing and returning the original object hierarchy.

math Library:

math.ceil(x)
: Return the ceiling of x, the smallest integer greater than or equal to x.

math.floor(x)
: Return the floor of x, the largest integer less than or equal to x.

math.exp(x)
: Return e raised to the power x, where e = 2.718281… is the base of natural logarithms.

math.cos(x)
: Return the cosine of x radians.

math.sin(x)
Return the sine of x radians.

math.tan(x)
Return the tangent of x radians.

math.pi
The mathematical constant π = 3.141592…, to available precision.

math.e
The mathematical constant e = 2.718281…, to available precision.

numpy Library:

numpy.array(list)
Generates an array. If the list or tuple is a 2D-list it generates a matrix. With a 1D-list it generates a vector

ndarray.ndim
the number of axes (dimensions) of the array.

ndarray.shape
the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with n rows and m columns, `shape` will be `(n,m)`. The length of the `shape` tuple is therefore the number of axes, `ndim`.

ndarray.size
the total number of elements of the array. This is equal to the product of the elements of `shape`.

ndarray.dtype
an object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. Additionally, NumPy provides types of its own. numpy.int32, numpy.int16, and numpy.float64 are some examples.

numpy.zeros(tuple)
creates a matrix with 0 using the tuple to define the number of lines and rows in the matrix

numpy.ones(tuple)
creates a matrix with 1 using the tuple to define the number of lines and rows in the matrix

numpy.arange(start,stop,step)
creates a vector starting at *start*, ending at *stop* using *step* between the values

numpy.ndarray.reshape(lines,rows)
reshapes a ndarray according to the values in *lines* and *rows*