Cheat Sheet Python: principal functions and methods Computer Programming in Python - 2021/2022 - Politecnico di Torino

Common Operations

`print(x1, x2, ..., xn, sep='␣', end='\n')`: `sep` is the separator character between values (default space), `end` is the final character (default next line).

`input(s)`: returns a string with the data entered from the keyboard (without `'\n'`). `s` is the initial message.

`range(i, j, k)`: creates a sequence of integers starting from `i` (included, default `0`), comes up to `j` (excluded, mandatory), with step `k` (default `1`).

For all containers `cont`:

`len(cont)`: returns the number of items.

`x in cont`: returns `True` if the item `x` is found in `cont`, `False` otherwise.

`sum(cont)`: returns the sum of the item values.

`max(cont)` / `min(cont)`: returns the max/min item.

`cont.clear()`: delete all items.

`sorted(cont)`: returns a new sorted list containing the items of `cont`. It supports all advanced options of list`.sort()`.

For all sequences `seq`:

`seq.count(x)`: returns how many times `x` is found in `seq`.

`seq[i]`: returns the index item `i` (`i<len(seq)`, otherwise `IndexError`). If `i<0`, starts from the bottom.

`seq[i:j]`: returns a subsequence with consecutive elements of `seq`, from the position `i` (included, default=`0`) up to the position `j` (excluded, default=`len(seq)`).

`seq[i:j:k]`: uses `k` as "step" to select items. If `k<0` and `i>j` goes backwards.

Math

`abs(a)` $= |a|$: returns the absolute value of `a`.

`round(a)`, `round(a, n)`: rounds the value of `a` to the nearest integer or to `n` decimal digits.

`floor(a)`: lower integer, `ceil(a)`: upper integer, `trunc(a)`: eliminates fractional part.

import math ↘

`math.sin(a)`, `cos(a)`, `tan(a)`, `exp(a)`, `log(a)`, `sqrt(a)`. They can raise `ValueError`

`math.isclose(a, b, rel_tol, abs_tol)`: returns `True` if `|a - b|` is less than or equal to `rel_tol` (relative tolerance) or `abs_tol` (absolute tolerance).

import random ↘

`random.random()`: returns a random number float in the range `[0,1)`.

`random.randint(i, j)`: returns a random integer between `i` and `j` (including extremes).

`random.uniform(a, b)`: returns a random real number between `a` e `b` (including extremes).

`random.choice(seq)`: returns any element of the sequence `seq`.

`random.shuffle(seq)`: shuffles the elements of the sequence in a random order `seq`.

String

`int(s)`: convert `s` in integer. Exception: `ValueError`.

`float(s)`: convert `s` in float. Exception: `ValueError`.

`str(x)`: convert `x` in string.

`ord(s)`: returns Unicode (integer) code of `s[0]`.

`chr(i)`: returns character corresponding to Unicode code `i`. Exception: `ValueError`.

`s+s1`: creates and returns a new string by concatenating two strings.

`s*n`:creates and returns a new string by concatenating `n` times the same string.

`s.lower()` / `s.upper()`: returns the lowercase / uppercase version of `s`.

`s.replace(s1, s2)` / `s.replace(s1, s2, n)`: returns a new version of `s` where each occurrence of `s1` is replaced by `s2`. If `n` is provided, replaces at most `n` occurrences.

`s.lstrip()` / `s.lstrip(s1)`: returns a new version of `s` where whitespace characters (spaces, tabs, newlines) are stripped from the beginning of `s`. If `s1` is provided, characters in it are deleted instead of whitespace characters.

`s.rstrip()` / `s.rstrip(s1)`: works as `lstrip`, but the characters are stripped from the end of `s`.

`s.strip()` / `s.strip(s1)`: works as `lstrip`, but the characters are deleted both at the beginning and at the end.

`s1 in s`:returns `True` if `s` contains `s1` as substring, otherwise `False`.

`s.count(s1)`: returns the number of non-overlapping occurrences of `s1` in `s`.

`s.startswith(s1)` / `s.endswith(s1)`: returns `True` if `s` starts/ends with `s1`, otherwise `False`.

`s.find(s1)` / `s.find(s1, i, j)`: returns the first index of `s` where an occurrence of `s1` begins, or `-1` if there is not. If any `i` and `j`, search in `s[i:j]`.

`s.index(s1)` / `s.index(s1, i, j)`: works as `find`, but if not present it raises `ValueError`.

`s.isalnum()`: returns `True` if `s` contains only letters or numbers and has at least one character, otherwise `False`.

`s.isalpha()`: returns `True` if `s` contains only letters and has at least one character, otherwise `False`.

`s.isdigit()`: returns `True` if `s` contains only digits and has at least one character, otherwise `False`.

`s.islower()` / `s.isupper()`: returns `True` if `s` it contains only lowercase/uppercase letters and has at least one character, otherwise `False`.

`s.isspace()`: returns `True` if `s` contains only whites-

pace characters (spaces, tabs and newlines) and has at least one character, otherwise `False`.

From strings to lists and vice versa:

`s.split(sep, maxsplit=n)`: returns a list of substrings obtained by splitting `s` at each occurrence of the string `sep` (separator). If `sep` is omitted, by default it is a sequence of whitespace characters. If `maxsplit` is specified, they will be done to the maximum `n` separations starting from the left (the list will have at most `n+1` items).

`s.rsplit(sep, maxsplit=n)`: works as `split`, but splits `s` starting from the right.

`s.splitlines()`: works as `split`, but uses `'\n'` as separator. Thus, split `s` in a list containing the single lines of text present in `s`.

`s.join(l)`: returns a single string containing all elements of `l` (which must be a list of strings) separated by the `s` separator.

Formatted strings `f'{x:fmt}'`

`x` is any variable or expression. `fmt` are *formatting codes*, which can contain:

`< ^ >`: left, centered, right alignment

width: number indicating how many characters in total the value must occupy. Default: just enough.

.precision: number of decimal digits (if float) or maximum number of characters (if not numeric).

format: `s` string, `d` integer, `f` real number, `g` real number in scientific notation.

Example: `f'{n:3d}␣{a:5.3f}␣{s:>25s}'`

List

`[]`: creates and returns a new empty list.

`[x1, ..., xn]`: returns a new list with the supplied items.

`list(cont)`: returns a *new* list containing all the elements of the container `cont`.

`l * n`: returns a new list by replicating the items of `l` for `n` times.

`l + l1`: returns a new list by concatenating the elements of `l` with `l1`.

`l == l1`: returns `True` if the two lists contain the same elements, in the same order, otherwise `False`.

`l.pop()`: removes the last element and returns it.

`l.pop(i)`: removes the item at the location `i` and returns it. The following items are shifted back one place.

`l.insert(i, x)`: insert `x` in the position `i` in `l`. Items from that position on are shifted forward one place.

`l.append(x)`: append `x` at the end of the list `l`.

`l.count(x)`: returns the number of occurrences of `x` in `l`

`l.index(x)`: returns the position of the first occurrence of `x` in `l`. The item must be present in the list, otherwise it raises `ValueError`.

`l.index(x, i, j)`: returns the position of the first occurrence of `x` in the list portion `l[i:j]`. The returned position is referenced from the beginning of the list. If not found, raise `ValueError`.

`l.remove(x)`: removes the item of value`x` from the list and shifts all the elements that follow it back one place. The item must be present in the list, otherwise it raises `ValueError`.

`l.extend(l1)`: adds all items of the list`l1` to the list `l`.

`l.reverse()`: reverses the order of the elements in the list `l`.

`l.copy()` o `list(l)`: returns a new list, copy of the list `l`.

`l.sort(reverse=False)`: sorts the items in the list from smallest to largest. If `reverse=True`is specified, sort in reverse order.

`enumerate(l)`: returns a list of type tuples `[(index1, val1), (index2, val2), ...]`, allowing to iterate simultaneously on indices and values of `l`.

from operator import itemgetter ↘

`l.sort(key=itemgetter('k'))`: sort a list of *dictionaries* based on the value of the keyed field `k`.

`l.sort(key=itemgetter(n))`: sort a list of *lists* or *tuples* based on the value of the index element `n`. Also useful when the list `l` is the result of the function `enumerate()` or `dict.items()`.

`max/min(l, key=itemgetter('k'))` : in a list of *dictionaries*, returns the item whose keyed field value`k` is max/min.

`max/min(l, key=itemgetter(n))`: in a list of *lists* or *tuples*, returns the element whose value of the index field `n` is max/min. Also useful when the list `l` is the result of the function `enumerate()` or `dict.items()`.

Note: `reverse` and `key` can be combined.

Set

`set()`: returns a new empty set.

`set(cont)`: returns a new collection that contains a copy of the `cont` (without duplicates).

`{x1, x2, ..., xn}`: returns a new set containing the indicated items (without duplicates).

`t.add(x)`: adds a new element to the collection `t`. If the element is already present, nothing happens.

`t.discard(x)`: deletes the item from the collection `t`. If the element does not belong to the collection, it has no effect.

`t.remove(x)`: wors as `discard`, but if the element is not present it raises `KeyError`.

`t == t1`: check whether the set `t` is equal to the set `t1`.

`t.issubset(t1)` or `t<=t1`: check if `t` $\subseteq$ `t1`.

`t.issuperset(t1)` o `t>=t1`: check if `t` $\supseteq$ `t1`.

`t.isdisjoint(t1)`: returns `True` if the intersection of the sets `t` and `t1` is null.

`t.union(t1)` o `t|t1`: returns a new set of `t` $\cup$ `t1`.

`t.intersection(t1)` o `t&t1`: returns a new set of `t` $\cap$ `t1`.

`t.difference(t1)` o `t-t1`: returns a new set that contains the items belonging to `t` but not to `t1`.

`t.symmetric_difference(t1)` o `t^t1`: returns a new set that contains the elements available in only one of the sets and not in both (x-or).

`t.copy()` o `set(t)`: returns a copy of the set `t`.

Dictionary

`k` = key: string, number, tuple

`dict()`: Returns a new empty dictionary.

`{}`: Returns a new empty dictionary.

`{k1:x1, ..., kn:xn}`: Returns a new dictionary containing the specified key/value pairs. `k in d`: returns `True` if the key `k` belongs to the dictionary `d`, otherwise `False`.

`d[k] = x`: Add a new key/value pair to the dictionary `d`, if `k` it is not already present, otherwise it modifies the value associated with the `k` key.

`d[k]`: returns the value associated with the `k` key, if it exists in `d`, otherwise raises `KeyError`.

`d.get(k, x)`: returns the value associated with the `k` key, if it exists in `d`, otherwise it returns the default value `x`.

`d.pop(k)`: delete from `d` the `k` key and the value associated with it; if not present, it raises `KeyError`. Returns the deleted value.

`d.items()`: returns a list[a] of tuples `(k,x)` of all items of `d`, in order of insertion.

`d.values()`:returns a list[a] containing all the values in `d`.

`d.keys()`: returns a list[a] with the dictionary keys, in order of insertion.

`sorted(d)`: returns an ordered list of dictionary keys.

`sorted(d.items())`: returns a list, sorted by key, of tuples `(k,x)` of items belonging to `d`.

`d.copy()` or `dict(d)`: returns a copy of the dictionary.

[a] to be precise, returns a *view*, which can be converted to a list with `list(...)` or which can be iterated with a `for...in` loop

import copy ↘

`copy.copy(x)`: Returns a simple ('shallow') copy of `x`. It builds a new container and inserts references to the values that were present in the original (`x`).

`copy.deepcopy(x)`: Returns a ('deep') copy of `x`. Build a new container and insert a new **copy** of the objects that were present in the original (`x`) (and so on with the objects contained in them).

Files

`f = open(s, mode, encoding='utf-8')`: opens the file named `s`. `mode`: `'r'` read, `'w'` write. Returns a "file object" `f`. Exceptions: `FileNotFoundError` if the file doesn't exist, `OSError` in general.

`f.close()`: closes the file `f`.

`f.readline()`: returns a string containing characters read from the file `f` until the next `'\n'` (included). Returns `""` if the end of the file has been reached.

`f.read(num)`: returns a string containing (at most) `num` characters read from file `f`. Given no arguments, it returns the entire file as a single string.

`f.readlines()`: returns the contents of the entire file as a list of strings, one string per line.

`f.write(s)`: writes `s` to file `f`. *Note*: it does not automatically add the newline character `'\n'`.

`print(..., file=f)`: like `print`, but it writes to file `f` rather than to the screen.

import csv ↘

`csv.reader(f)`: returns a 'CSV reader' object, which can be iterated upon with a `for` cycle, returning for each iteration a list containing the fields in the next line of file `f`.

`csv.DictReader(f, fieldnames=[...])`: returns a 'CSV dictionary reader' object, which can be iterated upon with a `for` cycle, returning for each iteration a dictionary having as values the fields in the next line of file `f` and as keys the elements of `fieldnames` (if omitted, the keys are read from the first line of the file).

`csv.writer(f)`: returns a 'CSV writer' object for file `f`, open for writing. Data may be written one line at a time using the method `writerow(one_record)` or `writerows(all_records)`.

Note: CSV files should always be opened using the option `newline=''` in the `open` function.

Common exception types

`ValueError`: passing an invalid value to a function (e.g. `math.sqrt(-1)`).

`IndexError`: trying to access a list or string outside of the allowed indices (e.g. `l[len(l)]`).

`KeyError`: trying to access a dictionary using a non-existent key.

`OSError` (or `IOError`): input-output errors, typically during file handling operations, such as `FileNotFoundError`, `PermissionError`, `FileExistsError`.

Legend (argument/object types)

`s`, `s1`: string
`a`, `b`, `c`, `...`: integer or float
`i`, `j`, `k`, `n`: integer
`x`: any
`l`, `l1`: list
`d`: dictionary
`t`, `t1`: set
`seq`: sequence (list, tuple, string)
`cont`: container (list, tuple, string, set, dict)

ver. 2.5 - last updated 10/01/2022