essentials-MARKDOWN

January 31, 2018

1 LIST METHODS

- list1.append(element)
- list1.extend(list2)
- list.insert(index, element)
- list1.index(element)... returns its smallest/first position
- list.count(element)... can take in single elements, lists and tuples
- list.remove(element)
- list.pop(index)... optional index (can be -ve)
- list.reverse()
- list.sort(key=..., reverse=...)

2 DICTIONARY METHODS

- dict.fromkeys(seq [, value])
- dict.items()
- dict.keys()
- dict.values()
- del[dict[key]]
- dict.update([other])
- dict.popitem()
- dict.pop(key[, default])
- dict.get(key[, value])
- dict.setdefault(key[, def_val])

3 SET METHODS

- A.difference(B) or "-", and A.difference_update(B)
- A.intersection(*other_sets)... can use "&" operator,
- A.symmetric_difference(B)... can also use "^" operator,
- A.union(*other_sets)... can also use the pipe operator "|"
- discard(x)
- remove(element)
- add(element)
- pop()

4 STRING METHODS

- capitalize()
- title()
- count(substring, start=..., end=...)
- startswith(prefix[, start[, end]]), str.endswith(suffix[, start[, end]])
- str.find(sub[, start[, end]]), rfind(sub[, start[, end]])
- index(sub[, start[, end]]), rindex(sub[, start[, end]])
- isalnum(), isalpha(), isdigit(), isnumeric()
- join()
- ljust(width[, fillchar]), rjust(width[, fillchar])
- lower(), upper(), swapcase()
- strip([chars]), lstrip([chars]), rstrip([chars])
- partition(separator)
- maketrans(x[, y[, z]]) rpartition(separator)
- translate(table)
- replace(old, new [, count])
- split([separator [, maxsplit]]) rsplit([separator [, maxsplit]])
- splitlines([keepends])

5 BUILT-IN FUNCTIONS

https://www.programiz.com/python-programming/methods/built-in

- chr(i)
- complex([real[, imag]])
- delattr(object, name)
- dict(**kwarg)(mapping,* *kwarg)(iterable, **kwarg) dir([object])

- divmod(x, y)
- enumerate(iterable, start=0)
- filter(function, iterable)
- float([x])
- format(value[, format_spec])
- frozenset([iterable])
- getattr(object, name[, default])
- hasattr(object, name)
- id(object)
- int(x=0, base=10)
- iter(object[, sentinel])
- map(function, iterable, ...)
- max(iterable, *iterables[,key, default])(arg1, arg2,* args[, key])
- min(iterable, *iterables[,key, default])(arg1, arg2,* args[, key])
- next(iterator, default)
- o = object()
- ord(c)
- pow(x, y[, z])
- repr(obj)

- reversed(seq)
- round(number[, ndigits])
- setattr(object, name, value)
- slice(start, stop, step)
- sorted(iterable[, key][, reverse])
- sum(iterable, start)
- type(object) (name, bases, dict)
- zip(*iterables)

6 ITERTOOLS

Itertools produces lazy iterables. Lazy iterables are objects. And… they only yield results one at a time, and only when u ask for them. Hence the need to call next(itertool object)... or to list(itertool object)

- itertools.accumulate(iterable[, func])
- itertools.chain(*iterables)
- itertools.combinations(iterable, r)
- itertools.combinations_with_replacement(iterable, r)
- itertools.compress(data, selectors)
- itertools.count(start=0, step=1)
- itertools.cycle(iterable)
- itertools.dropwhile(predicate, iterable)
- itertools.filterfalse(predicate, iterable)
- itertools.groupby(iterable, key=None)
- itertools.isslice(iterable, stop)
- itertools.isslice(iterable, start, stop[, step])
- itertools.permutations(iterable, r=None)
- itertools.product(*iterables, repeat=1)
- itertools.repeat(object[, times])
- itertools.starmap(function, iterable)
- itertools.takewhile(predicate, iterable)
- itertools.tee(iterable, n=2)ű
- itertools.zip_longest(*iterables, fillvalue=None)

7 COLLECTIONS

namedtuple(typename, field_names, *, verbose=False, rename=False, module=None) - classmethod somenamedtuple._make(iterable) - somenamedtuple._asdict() - somenamedtuple._replace(kwargs) - somenamedtuple._source - somenamedtuple._fields - namedtuple.count() - namedtuple.index()**

8 class collections.deque([iterable[, maxlen]])

- deque.append(x)
- deque.appendleft(x)

- deque.clear()
- deque.copy()
- deque.count(x)
- deque.extend(iterable)
- deque.extendleft(iterable)
- deque.index(x[, start[, stop]])
- deque.insert(i, x)
- deque.pop()
- deque.popleft()
- deque.remove(value)
- deque.reverse()
- deque.rotate(n)
- deque.maxlen

9 class collections.Counter([iterable-or-mapping])

- c = Counter(list)
- c.elements()
- c.most_common()
- c.subtract()
- c+c
- sum(c.values()) # total of all counts
- c.clear() # reset all counts
- list(c) # list unique elements
- set(c) # convert to a set
- dict(c) # convert to a regular dictionary
- c.items() # convert to a list of (elem, cnt) pairs
- Counter(dict(list_of_pairs)) # convert from a list of (elem, cnt) pairs
- c.most_common()[:-n-1:-1] # n least common elements
- +c # remove zero and negative counts