ΠΕΡΙΕΧΟΜΕΝΑ:

1. **Counter**
2. **defaultdict και OrderedDict**
3. **ChainMap**
4. **deque**
5. **namedtuple() και user***

Αγγελική Γ. - Σταυριανή Γ.

Ασημένιος Χορηγός Μαθήματος

Νικός Κώνστας

Ασημένιος Χορηγός Μαθήματος

ΜΑΘΗΜΑ 2.2: Το module collections

1. Η κλάση Counter

- Το **module collections** περιέχει κλάσεις - συλλογές που μοντελοποιούν δομές δεδομένων, είτε νέες, είτε επεκτάσεις των built-in (dict, list, tuple κ.λπ.)

Η κλάση Counter (Μετρητής Εμφανίσεων Στοιχείου σε Δομή):

- Είναι υποκλάση του λεξικού:
  - αρχικοποιείται με μία δομή δεδομένων που πρέπει να περιέχει hashable αντικείμενα (immutable, αντικ/να δικών μας κλάσεων).
  - κλειδιά του λεξικού είναι τα διακριτά στοιχεία της δομής, με τιμή κλειδιού το πλήθος εμφανίσεων του κλειδιού (δηλ. μετράει πόσες φορές εμφανίζεται κάθε στοιχείο στη δομή)

Παράδειγμα 1: counter\counter_init.py

```
from collections import Counter

l = [1, 2, 3, 2]
cl = Counter(l)
print(cl)

phrase = "Winter is coming"
cphrase = Counter(phrase)
print(cphrase, cphrase["i"])

t = [[1, 2], [1]]
ct = Counter(t) # error: elems=>unhashable
```

Μέθοδοι της κλάσης Counter:

Μέθοδος	Επεξήγηση
most_common(n)	Επιστρέφει λίστα από τα n tuples (key, value) με τα πιο συχνά κλειδιά
elements()	Επιστρέφει iterator όπου κάθε κλειδί εμφανίζεται τόσες φορές, όσες και η τιμή του (εφόσον είναι >0)
subtract(dict)	Αφαιρεί από την τιμή κάθε κλειδίου, την τιμή που περιέχεται στο ίδιο κλειδί στο dict

- Αφού κληρονομεί το λεξικό:
  - περιέχει όλες τις μεθόδους που είδαμε στο μάθημα 8, π.χ. pop(k), clear(), keys(), values(), items()…
  - διαφέρουν στο ότι αν ανασύρουμε την τιμή κλειδιού που δεν υπάρχει (πχ cnt[key]) τότε η Counter θα μας απαντήσει 0 (ενώ το dict προκαλεί KeyError)

Παράδειγμα 2: counter\counter_methods

```
mystr = "..."
words = mystr.replace(",","").replace(".","").lower().split()
word_count = Counter(words)
print(word_count)
print(word_count.most_common(2))
word_count.subtract({"the": 2})
print(word_count)
```

ΜΑΘΗΜΑ 2.2: Το module collections — 2. defaultdict και OrderedDict

- Η **κλάση defaultdict** είναι υποκλάση του λεξικού
- δίνει αυτόματα μία προκαθορισμένη τιμή, όταν γίνεται πρόσβαση σε κλειδί που δεν υπάρχει στο λεξικό.

πχ η εντολή:

```
print(mydict[“key”])
```

- (εφόσον το «key» δεν υπάρχει στο λεξικό)
- σε ένα κανονικό λεξικό θα χτυπούσε KeyError
- σε ένα defaultdict θα πρόσθετε το κλειδί key με τη default τιμή και θα τύπωνε την τιμή αυτή.

- **Κατασκευάζουμε ένα defaultdict διοχετεύοντας στον κατασκευαστή ένα callable** (π.χ. συνάρτηση ή κατασκευαστή κλάσης που επιστρέφει το αντικείμενο, με το οποίο γίνεται η αρχικοποίηση της τιμής όταν δεν υπάρχει κλειδί)

Παράδειγμα 3: dict_variations/default_dict.py

```
def func():
    return "no value"
d1 = defaultdict(func)
print(d1["a"])
```

```
s = [('yellow', 1), ('blue', 2), ('yellow', 3), ('blue', 4), ('red', 1)]
d4 = defaultdict(list)
for k, v in s:
    d4[k].append(v)
print(d4)
```

- Η **κλάση OrderedDict** είναι υποκλάση του λεξικού
- διατηρεί τα ζεύγη ταξινομημένα με βάση τη σειρά εισαγωγής

Παράδειγμα 4: dict_variations/ordered_dict.py

```
d = OrderedDict()
d[1] = 11
d[2] = 12
d[3] = 13
d[0] = 10
for k,v in d.items():
    print(f"{k}: {d[k]}")
print(d.popitem())
```

Η συμπεριφορά είναι LIFO (Last-In First-Out)

- Αν θέσουμε το όρισμα με λέξη κλειδί στην popitem(): Last=False, τότε έχει συμπεριφορά FIFO (First-In First-Out)

Παράδειγμα 5: dict_variations/ordered_dict2.py

```
d = OrderedDict()
...
for k,v in d.items():
    print(f"{k}: {d[k]}")

print(d.popitem(last=False))
```

- Η κλάση OrderedDict είναι κάπως παρωχημένη, αφού μετά την Python 3.8, η ίδια ταξινόμηση ενσωματώθηκε στο απλό λεξικό.

3. ChainMap

Η **κλάση ChainMap** περιτυλίσσει πολλά λεξικά

- και δίνει μία διεπαφή, ώστε να έχουμε πρόσβαση σε ένα κλειδί κάποιου από αυτά λεξικά με έναν εύκολο τρόπο.

Κατασκευάζουμε ένα ChainMap:

Κατασκευαστής	Επεξήγηση
ChainMap(*args)	ChainMap με αυθαίρετο πλήθος από λεξικά (args)

- Εσωτερικά διατηρείται ως μία λίστα από λεξικά, στην οποία έχουμε πρόσβαση με το μέλος: **maps**
- Η ανασυρση της τιμής ενός κλειδιού, θα αναζητήσει σε όλα τα λεξικά και θα επιστρέψει την τιμή από το λεξικό που βρίσκεται το κλειδί (αν υπάρχει σε πολλά, θα επιστρέψει την τιμή από το πρώτο λεξικό που θα βρει την τιμή, με βάση τη διάταξή τους).

Παράδειγμα 5: chain_map/constructor.py

```
from collections import ChainMap

dict1 = {"a": 1, "b": 2}
dict2 = {"c": 3, "d": 4}
dict3 = {"e": 5, "a": 6}

cm = ChainMap(dict1, dict2, dict3)
print(cm)

print(cm["b"])
print(cm["a"])
```

Μέλη - Μέθοδοι του ChainMap:

Μέλος	Επεξήγηση
maps	Λίστα με όλα τα λεξικά (τροποποιήσιμη - καθορίζει τη σειρά εξερεύνησης των λεξικών)

Μέθοδος	Επεξήγηση
new_child(map=None)	Επιστρέφει νέο ChainMap που περιέχει το λεξικό map, ακολουθούμενο από όλα τα λεξικά του αρχικού ChainMap Αν δεν διοχετευτεί όρισμα, τότε το νέο λεξικό είναι κενό.

Ορίζεται επίσης το property **parents** (επιστρέφει όλα τα λεξικά, εκτός από το 1ο - βλ. βίντεο)

Παράδειγμα 6: chain_map/members.py

```
cm = ChainMap(dict1, dict2, dict3)
print(cm.maps)
newcm = cm.parents
print(newcm)

ccm = cm.new_child({"f": 1})
print(ccm)
ccm.maps[0], ccm.maps[1] = ccm.maps[1], ccm.maps[0]
print(ccm)
```

Η **κλάση deque** μοντελοποιεί μια ουρά με δύο άκρα (double-ended queue)

- κατασκευασμένη ώστε να κάνει πολύ γρήγορα (σταθερός χρόνος - O(1)) εισαγωγές - διαγραφές στα άκρα της.

Κατασκευάζουμε ένα deque:

Κατασκευαστής	Επεξήγηση
deque ([iterable[,maxlen]])	Προαιρετική αρχικοποίηση με το iterable. Αν δεν καθοριστεί μέγιστο μήκος, είναι πρακτικά απεριοριστου μεγέθους, αλλιώς εισαγωγές σε γεμάτη ουρά προκαλούν απομάκρυνση από το άλλο άκρο της ουράς με δύο άκρα.

- **βασικές μέθοδοι:**

Μέθοδος	Επεξήγηση
append(x)	Προσθήκη του x στο τέλος (δεξιά)
appendleft(x)	Προσθήκη του x στην αρχή (αριστερά)
pop()	Επιστροφή και απομάκρυνση από το τέλος
popleft()	Επιστροφή και απομάκρυνση από την αρχή

Δευτερεύουσες μέθοδοι:

Μέθοδος	Επεξήγηση
clear()	Διαγράφει όλα τα στοιχεία του deque
copy()	Δημιουργεί shallow copy του deque (mod.2.1)
count(x)	Μετράει πόσα στοιχεία είναι ίσα με το x
extend(iterable)	Επεκτείνει το deque στα δεξιά με τα στοιχεία του iterable
extendleft(iterable)	Επεκτείνει το deque στα αριστερά με τα στοιχεία του iterable
index(x[,start[,stop]])	Επιστρέφει τη θέση του x στο deque
remove(x)	Αφαιρεί την πρώτη εμφάνιση του x
reverse()	Αντιστροφή του deque
rotate(n=1)	Περιστροφή των στοιχείων κατά n θέσεις

και έχουν υπερφορτωθεί οι len(deque), reversed(deque), copy.copy(deque), copy.deepcopy(deque), ο τελεστής in και πρόσβαση με αγκύλες όπως στις λίστες

Παράδειγμα 7: doubleEndedQueue/exercise12_lesson17.py

```
class Queue:
    def __init__(self):
        self.data = deque()
    def enqueue(self, elem):
        self.data.append(elem)
    def dequeue(self):
        if len(self.data) == 0:
            return None
        else:
            return self.data.popleft()
```

Παράδειγμα 8: doubleEndedQueue/methods.py

```
class Stack:
    def __init__(self):
        self.stack = deque()

    def push(self, element):
        self.stack.append(element)
    def pop(self):
        if len(self.stack) == 0:
            return None
        else:
            return self.stack.pop()
```

ΜΑΘΗΜΑ 2.2: Το module collections — 5. namedtuple() και User*

Η δομή **named tuple** είναι μια επέκταση του tuple:

- δίνει τη δυνατότητα να έχουμε πρόσβαση στα μέλη της, όχι μόνο με το συνήθη τρόπο (indexes π.χ. t[0]), αλλά και με τρόπο αντίστοιχο με τα μέλη μίας κλάσης.

Κατασκευή μέσω της factory method:

factory method	Επεξήγηση
namedtuple (“name”, “at1 at2 …”)	Κατασκευάζει ένα named tuple (υποκλάση της tuple) στο οποίο οι διαδοχικές θέσεις είναι προσβάσιμες ως name.at1, name.at2, …

Παράδειγμα 9: named_tuple/construct_access.py

```
from collections import namedtuple
Point = namedtuple("Point", "x y")
p = Point(1.4, 2.8)
print(p[0], p[1])
print(p.x, p.y)
```

Παράδειγμα 10: named_tuple/query

```
query = "SELECT Code, Name, Population FROM country"
cursor.execute(query)
rows = cursor.fetchall()
print(rows)
Country = namedtuple("Country", "Code Name Population")
c = Country(*rows[0])
```

Μέθοδοι (ενός αντικειμένου namedtuple):

Μέθοδος	Επεξήγηση
_make(iterable)	Μετατρέπει το iterable σε named tuple
_asdict()	Επιστρέφει αναπαράσταση ως λεξικό
_replace(**kwargs)	Αντικαθιστά τα keyword arguments με τις νέες τιμές και επιστρέφει νέο αντικ/νο

και ορίζεται επίσης το μέλος:

Μέλος	Επεξήγηση
_fields	Επιστρέφει tuple με τα ονόματα των θέσεων

Ενώ μπορούμε:

- να μετατρέψουμε λεξικό σε named tuple: namedtuple(**dict)

Παράδειγμα 11: named_tuple/methods.py

```
Point = namedtuple("Point", "x y")
p = Point(1.1, 2.8)
print(p._make([1, 1]))
print(p._asdict())
print(p._replace(x=3.3))
print(p._fields)
```

Στο module collections:

- Ορίζονται επίσης οι κλάσεις UserDict, UserList, UserString, οι οποίες ορίζουν επεκτάσεις χρήστη στις πρωτογενείς δομές.
- Αντίστοιχη συμπεριφορά μπορούμε να πετύχουμε με την κληρονομικότητα, οπότε δεν θα τις μελετήσουμε εδώ.