# Πλειάδες (Tuples)

Οι πλειάδες είναι μια δομή δεδομένων στην Python που χρησιμοποιείται για την αποθήκευση συλλογών στοιχείων. Έχουν κάποιες ιδιαίτερες ιδιότητες που τις διαφοροποιούν από άλλες δομές δεδομένων:

## Βασικά Χαρακτηριστικά:

1. **Αμετάβλητες (Immutable)**: Μετά τη δημιουργία τους, δεν μπορούμε να αλλάξουμε, να προσθέσουμε ή να αφαιρέσουμε στοιχεία.
2. **Διατεταγμένες (Ordered)**: Τα στοιχεία διατηρούν τη σειρά εισαγωγής τους.
3. **Επιτρέπουν διπλότυπα**: Μπορούν να περιέχουν πολλαπλές εμφανίσεις του ίδιου στοιχείου.
4. **Ετερογενείς**: Μπορούν να περιέχουν στοιχεία διαφορετικών τύπων δεδομένων.

## Δημιουργία και Χρήση Πλειάδων:

```
# Δημιουργία πλειάδας
simple_tuple = (1, 2, 3, 4, 5)
mixed_tuple = (1, "δύο", 3.0, True)

# Πρόσβαση σε στοιχεία με ευρετήριο (index)
print(simple_tuple[0])  # 1
print(mixed_tuple[1])   # "δύο"

# Μήκος πλειάδας
print(len(simple_tuple))  # 5

# Έλεγχος ύπαρξης στοιχείου
print(3 in simple_tuple)  # True
```

## Σύνθετοι Τύποι σε Πλειάδες:

Οι πλειάδες (καθώς και οι περισσότερες συλλογές δεδομένων στην Python) μπορούν να περιέχουν σύνθετους τύπους δεδομένων, συμπεριλαμβανομένων άλλων πλειάδων, λιστών, λεξικών, κ.λπ.

```
# Πλειάδα με σύνθετους τύπους
complex_tuple = (
    [1, 2, 3],           # Λίστα
    {"a": 1, "b": 2},    # Λεξικό
    (4, 5, 6),           # Εμφωλευμένη πλειάδα
    {7, 8, 9}            # Σύνολο <- θα το μάθουμε αργότερα
)

print(complex_tuple[0][1])    # 2 (δεύτερο στοιχείο της λίστας)
print(complex_tuple[1]["a"])  # 1 (τιμή του κλειδιού "a" στο λεξικό)
print(complex_tuple[2][0])    # 4 (πρώτο στοιχείο της εμφωλευμένης πλειάδας)
print(7 in complex_tuple[3])  # True (έλεγχος ύπαρξης στο σύνολο)
```

In [ ]:

# Indexing και Slicing σε Συλλογές Δεδομένων

Στην Python, το indexing και το slicing είναι δύο βασικές τεχνικές για την πρόσβαση σε στοιχεία συλλογών δεδομένων, όπως λίστες, tuples και συμβολοσειρές.

## Indexing (ευρετηρίαση)

Το indexing επιτρέπει την πρόσβαση σε μεμονωμένα στοιχεία μιας συλλογής. Είναι σημαντικό να θυμόμαστε ότι στην Python, όπως και σε πολλές άλλες γλώσσες προγραμματισμού, η αρίθμηση των στοιχείων ξεκινά πάντα από το 0.

```
# Δημιουργία ενός tuple
fruits = ("μήλο", "μπανάνα", "πορτοκάλι", "φράουλα", "ακτινίδιο")

print(f"Το tuple μας: {fruits}")

# Πρόσβαση στο πρώτο στοιχείο (index 0)
print(f"Πρώτο φρούτο: {fruits[0]}")  # Εκτύπωση: μήλο

# Πρόσβαση στο τρίτο στοιχείο (index 2)
print(f"Τρίτο φρούτο: {fruits[2]}")  # Εκτύπωση: πορτοκάλι

# Αρνητικό indexing: πρόσβαση από το τέλος
print(f"Τελευταίο φρούτο: {fruits[-1]}")  # Εκτύπωση: ακτινίδιο
print(f"Προτελευταίο φρούτο: {fruits[-2]}")  # Εκτύπωση: φράουλα

# Προσπάθεια πρόσβασης σε index εκτός ορίων θα προκαλέσει σφάλμα
# print(fruits[5])  # Αυτό θα προκαλέσει IndexError
```

## Slicing (τεμαχισμός)

Το slicing επιτρέπει την εξαγωγή ενός εύρους στοιχείων από μια συλλογή. Η σύνταξη του slicing είναι `[αρχή:τέλος:βήμα]`, όπου:

- `αρχή`: το index από το οποίο ξεκινά το slice (συμπεριλαμβάνεται)
- `τέλος`: το index στο οποίο τελειώνει το slice (δεν συμπεριλαμβάνεται)
- `βήμα`: (προαιρετικό) πόσα στοιχεία θα προχωράει κάθε φορά

Είναι σημαντικό να θυμόμαστε ότι το δεξί άκρο (τέλος) δεν συμπεριλαμβάνεται στο αποτέλεσμα.

```
# Χρησιμοποιώντας το ίδιο tuple fruits

# Slice από το δεύτερο μέχρι το τέταρτο στοιχείο (χωρίς να συμπεριλαμβάνεται
το τέταρτο)
print(f"Δεύτερο και τρίτο φρούτο: {fruits[1:3]}")  # Εκτύπωση: ('μπανάνα',
'πορτοκάλι')

# Slice από την αρχή μέχρι το τρίτο στοιχείο
print(f"Πρώτα τρία φρούτα: {fruits[:3]}")  # Εκτύπωση: ('μήλο', 'μπανάνα',
'πορτοκάλι')

# Slice από το τρίτο στοιχείο μέχρι το τέλος
print(f"Τελευταία τρία φρούτα: {fruits[2:]}")  # Εκτύπωση: ('πορτοκάλι',
'φράουλα', 'ακτινίδιο')

# Slice με βήμα 2 (κάθε δεύτερο στοιχείο)
print(f"Κάθε δεύτερο φρούτο: {fruits[::2]}")  # Εκτύπωση: ('μήλο',
'πορτοκάλι', 'ακτινίδιο')

# Αρνητικό slicing (από το τέλος)
print(f"Τελευταία δύο φρούτα: {fruits[-2:]}")  # Εκτύπωση: ('φράουλα',
'ακτινίδιο')

# Αντιστροφή του tuple
```

```
print(f"Αντεστραμμένη λίστα φρούτων: {fruits[::-1]}")  # Εκτύπωση:
('ακτινίδιο', 'φράουλα', 'πορτοκάλι', 'μπανάνα', 'μήλο')
```

Σημειώσεις:

1. Το slicing λειτουργεί με τον ίδιο τρόπο σε λίστες, tuples και συμβολοσειρές.
2. Αν παραλείψουμε την `αρχή`, το slice ξεκινά από την αρχή της συλλογής.
3. Αν παραλείψουμε το `τέλος`, το slice συνεχίζει μέχρι το τέλος της συλλογής.
4. Αν χρησιμοποιήσουμε αρνητικό `βήμα`, η διάσχιση γίνεται από το τέλος προς την αρχή.

In [ ]:

# Λίστες

Οι λίστες είναι ένας από τους πιο ευέλικτους και συχνά χρησιμοποιούμενους τύπους δεδομένων στην Python. Γενικά, έχουν τις ίδιες ιδιότητες με τα tuples, **αλλά είναι μεταβλητές (mutable)**.

## Δημιουργία λίστας

```
# Δημιουργία λίστας
fruits_list = ["μήλο", "μπανάνα", "πορτοκάλι"]
print(fruits_list)
```

## Πρόσβαση σε στοιχεία - indexing (ακριβώς όπως και με τα tuples)

```
# Πρόσβαση σε στοιχεία με δείκτη (index)
print(fruits_list[0])  # Πρώτο στοιχείο
print(fruits_list[-1])  # Τελευταίο στοιχείο
```

## Τεμαχισμός - slicing (ακριβώς όπως και με τα tuples)

```
print(fruits_list[1:-1])  # ["μπανάνα"]
```

## Προσθήκη στοιχείων

```
# Προσθήκη στοιχείου στο τέλος
fruits_list.append("φράουλα")
print(fruits_list)

# Προσθήκη στοιχείου σε συγκεκριμένη θέση
fruits_list.insert(1, "κεράσι")
print(fruits_list)
```

## Αφαίρεση στοιχείων

```
# Αφαίρεση τελευταίου στοιχείου
last_fruit = fruits_list.pop()
print(f"Αφαιρέθηκε: {last_fruit}")
print(fruits_list)

# Αφαίρεση συγκεκριμένου στοιχείου
fruits_list.remove("μπανάνα")
print(fruits_list)
```

## Διαφορά μεταξύ Λίστας και Tuple

Οι λίστες και τα tuples είναι και τα δύο ακολουθίες στην Python, αλλά έχουν μερικές σημαντικές διαφορές:

1. **Μεταβλητότητα**: Οι λίστες είναι μεταβλητές (mutable), ενώ τα tuples είναι αμετάβλητα (immutable).
2. **Σύνταξη**: Οι λίστες χρησιμοποιούν αγκύλες `[]`, ενώ τα tuples χρησιμοποιούν παρενθέσεις `()`.
3. **Χρήση**: Οι λίστες χρησιμοποιούνται *ΣΥΝΗΘΩΣ* για ομοιογενείς ακολουθίες που μπορεί να αλλάξουν, ενώ τα tuples για ετερογενείς ακολουθίες που παραμένουν σταθερές.

```
# Λίστα
my_list = [1, 2, 3]
my_list[0] = 10  # Επιτρέπεται

# Tuple
my_tuple = (1, 2, 3)
# my_tuple[0] = 10  # Θα προκαλέσει σφάλμα
```

In [ ]:

# Επαναλήψεις

## For loop

Η `for` επανάληψη χρησιμοποιείται για να επαναλάβουμε μια ακολουθία (όπως λίστα, tuple, λεξικό, σύνολο ή συμβολοσειρά).

```
# Επανάληψη σε λίστα
for fruit in fruits:
    print(fruit)

# Επανάληψη με αρίθμηση
for index, fruit in enumerate(fruits):
    print(f"{index + 1}. {fruit}")
```

## While loop

Η `while` επανάληψη εκτελεί ένα σύνολο εντολών όσο μια συνθήκη είναι αληθής.

```
# Απλή while επανάληψη
count = 0
while count < 5:
    print(count)
    count += 1

# While με έλεγχο χρήστη
response = ""
while response.lower() != "quit":
    response = input("Πληκτρολογήστε κάτι (ή 'quit' για έξοδο): ")
    print(f"Πληκτρολογήσατε: {response}")
```

In [ ]:

# Συναρτήσεις

Οι συναρτήσεις είναι επαναχρησιμοποιήσιμα κομμάτια κώδικα που εκτελούν συγκεκριμένες εργασίες. Βοηθούν στην οργάνωση του κώδικα και στη μείωση της επανάληψης.

## Ορισμός Συνάρτησης

```
def greet(name):
    """Αυτή η συνάρτηση χαιρετά το άτομο που περνάει ως όρισμα"""
    print(f"Γεια σου, {name}!")

# Κλήση της συνάρτησης
greet("Μαρία")
```

## Ορίσματα Συναρτήσεων

Οι συναρτήσεις μπορούν να δέχονται διάφορους τύπους ορισμάτων:

1. **Υποχρεωτικά ορίσματα**:

```
def add(a, b):
    return a + b

result = add(5, 3)
print(result)  # Εκτύπωση: 8
```

2. **Προαιρετικά ορίσματα** (με προεπιλεγμένες τιμές):

```
def power(base, exponent=2):
    return base ** exponent

print(power(3))     # Εκτύπωση: 9 (3^2)
print(power(3, 3))  # Εκτύπωση: 27 (3^3)
```

3. **Αόριστος αριθμός ορισμάτων**:

```
def sum_all(*args):
    s=0
    for i in args:
        s+=i
    return s

print(sum_all(1, 2, 3, 4))  # Εκτύπωση: 10
```

## Επιστρεφόμενες Τιμές

Οι συναρτήσεις μπορούν να επιστρέφουν τιμές χρησιμοποιώντας τη λέξη-κλειδί `return`:

```
def calculate_area(radius):
    return 3.14 * radius ** 2

area = calculate_area(5)
print(f"Το εμβαδόν του κύκλου είναι {area}")
```

### Επιστροφή Πολλαπλών Τιμών

Όταν θέλουμε ενα επιστρέψουμε πολλές τιμές απο μια συνάρτηση, συνήθως χρησιμοποιούμε tuples.

```
def get_user_info():
    return ("Μαρία", 25, "Αθήνα")
```

```python
#εδώ χρησιμοποιούμε tuple unpacking ώστε να εκχωρίσουμε σε διαφορετικές μεταβλητές τα δεδομένα που υπάρχουν μέσα σε ένα tuple

name, age, city = get_user_info()
print(f"Η {name} είναι {age} ετών και ζει στην {city}.")
```

In [ ]:

## Η συνάρτηση range()

Η range() δημιουργεί μια ακολουθία αριθμών, χρήσιμη για βρόχους for.

```python
# range(stop)
for i in range(5):
    print(i)  # 0, 1, 2, 3, 4

# range(start, stop)
for i in range(2, 5):
    print(i)  # 2, 3, 4

# range(start, stop, step)
for i in range(0, 10, 2):
    print(i)  # 0, 2, 4, 6, 8

# Δημιουργία λίστας με range
numbers = list(range(1, 6))
print(numbers)  # [1, 2, 3, 4, 5]
```

### Συνδυασμός range() και for

```python
# Παράδειγμα: Υπολογισμός πολλαπλασιασμού
number = 5
for i in range(1, 11):
    print(f"{number} x {i} = {number * i}")
```

In [27]: