# Python Dictionaries (Λεξικά)

## Εισαγωγή

Τα dictionaries (λεξικά) είναι μία από τις πιο χρήσιμες δομές δεδομένων στην Python. Λειτουργούν όπως ένα πραγματικό λεξικό - κάθε λέξη (κλειδί/key) έχει έναν ορισμό (τιμή/value).

## Βασικά Χαρακτηριστικά

- Τα λεξικά δημιουργούνται με άγκιστρα `{}` (curly braces)
- Κάθε στοιχείο αποτελείται από ζεύγη κλειδιού-τιμής (key-value pairs)
- Τα κλειδιά πρέπει να είναι μοναδικά (unique)
- Τα κλειδιά πρέπει να είναι αμετάβλητα (immutable) - συνήθως strings, αριθμοί ή tuples

## Δημιουργία Λεξικού (Creating a Dictionary)

```
# Empty dictionary
student = {}

# Dictionary with data
student = {
    "name": "Maria",
    "age": 19,
    "course": "Python"
}
```

## Βασικές Λειτουργίες (Basic Operations)

### 1. Προσθήκη/Τροποποίηση Στοιχείων (Adding/Modifying Elements)

```
# Adding new element
student["grade"] = 8.5

# Modifying existing element
student["age"] = 20
```

### 2. Πρόσβαση σε Στοιχεία (Accessing Elements)

```
# Using the key
print(student["name"])     # Prints: Maria

# Using get method
# Safer method - doesn't raise an error if key doesn't exist
grade = student.get("grade", 0)  # Returns 0 if key doesn't exist
```

### 3. Διαγραφή Στοιχείων (Deleting Elements)

```
# Delete specific element
del student["course"]

# Clear entire dictionary
student.clear()
```

## Χρήσιμες Μέθοδοι (Useful Methods)

```
# List of all keys
keys = student.keys()

# List of all values
values = student.values()

# List of all key-value pairs
items = student.items()
```

## Επανάληψη σε Λεξικό (Iterating Over a Dictionary)

```
# Iterating over keys
for key in student:
    print(key)

# Iterating over key-value pairs
for key, value in student.items():
    print(f"{key}: {value}")
```

## Παράδειγμα Χρήσης (Usage Example)

```
# Creating a students dictionary
students = {
    "1": {
        "name": "George",
        "grades": [7, 8, 9]
    },
    "2": {
        "name": "Helen",
        "grades": [9, 9, 8]
    }
}

# Calculate average grade for each student
for student_id, data in students.items():
    average = sum(data["grades"]) / len(data["grades"])
    print(f"{data['name']} has average grade: {average}")
```

## Συμβουλές (Tips)

1. Χρησιμοποιήστε περιγραφικά κλειδιά (Use descriptive keys)
2. Προτιμήστε τη μέθοδο `.get()` για ασφαλή πρόσβαση (Prefer .get() for safe access)
3. Θυμηθείτε ότι τα κλειδιά πρέπει να είναι μοναδικά (Remember keys must be unique)
4. Τα λεξικά είναι ιδανικά για δεδομένα που θέλετε να προσπελάσετε με ένα μοναδικό αναγνωριστικό (Dictionaries are ideal for data you want to access with a unique identifier)

In [ ]:

In [ ]:

In [ ]:

# Lambda Functions στην Python (Ανώνυμες Συναρτήσεις)

Οι lambda functions (ή ανώνυμες συναρτήσεις) είναι μικρές, ανώνυμες συναρτήσεις που μπορούν να έχουν οποιονδήποτε αριθμό παραμέτρων αλλά μόνο μία έκφραση. Είναι χρήσιμες όταν χρειαζόμαστε μια απλή συνάρτηση για σύντομο χρονικό διάστημα.

## Βασική Σύνταξη

```
lambda arguments: expression
```

## Σύγκριση με Κανονικές Συναρτήσεις

```
# Regular function
def add(x, y):
    return x + y

# Equivalent lambda function
add = lambda x, y: x + y

# Usage
print(add(5, 3))  # Output: 8
```

## Απλά Παραδείγματα

```
# Square a number
square = lambda x: x**2
print(square(5))  # Output: 25

# Convert celsius to fahrenheit
to_fahrenheit = lambda celsius: (celsius * 9/5) + 32
print(to_fahrenheit(20))  # Output: 68.0

# Check if number is even
is_even = lambda x: x % 2 == 0
print(is_even(4))  # Output: True
```

## Χρήση με Ενσωματωμένες Συναρτήσεις

### 1. Sorted()

```
# Sorting a list of tuples by second element
pairs = [(1, 'one'), (3, 'three'), (2, 'two')]
sorted_pairs = sorted(pairs, key=lambda x: x[1])
print(sorted_pairs)  # Output: [(1, 'one'), (3, 'three'), (2, 'two')]

# Sorting a list of dictionaries
students = [
    {'name': 'John', 'grade': 85},
    {'name': 'Maria', 'grade': 92},
    {'name': 'Peter', 'grade': 78}
]
```

```python
# Sort by grade (ascending)
by_grade = sorted(students, key=lambda x: x['grade'])
print("\nSorted by grade:")
for student in by_grade:
    print(f"{student['name']}: {student['grade']}")

# Sort by name (alphabetically)
by_name = sorted(students, key=lambda x: x['name'])
print("\nSorted by name:")
for student in by_name:
    print(f"{student['name']}: {student['grade']}")

# Sort by grade (descending)
by_grade_desc = sorted(students, key=lambda x: x['grade'], reverse=True)
print("\nSorted by grade (descending):")
for student in by_grade_desc:
    print(f"{student['name']}: {student['grade']}")
```

### 2. Filter()

```python
# Filter even numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)  # Output: [2, 4, 6, 8, 10]

# Filter dictionaries
products = [
    {'name': 'laptop', 'price': 1000},
    {'name': 'mouse', 'price': 25},
    {'name': 'monitor', 'price': 300}
]
affordable = list(filter(lambda x: x['price'] < 500, products))
print(affordable)  # Output: [{'name': 'mouse', 'price': 25}, {'name':
'monitor', 'price': 300}]
```

### 3. Map()

```python
# Apply function to all elements
numbers = [1, 2, 3, 4, 5]
squares = list(map(lambda x: x**2, numbers))
print(squares)  # Output: [1, 4, 9, 16, 25]

# Convert temperatures
celsius = [0, 10, 20, 30, 40]
fahrenheit = list(map(lambda x: (x * 9/5) + 32, celsius))
print(fahrenheit)  # Output: [32.0, 50.0, 68.0, 86.0, 104.0]
```

## Προχωρημένα Παραδείγματα

### 1. Σύνθετη Ταξινόμηση

```python
# Multi-level sorting
students = [
    {'name': 'John', 'grade': 85, 'age': 20},
    {'name': 'Maria', 'grade': 92, 'age': 19},
    {'name': 'Peter', 'grade': 85, 'age': 21},
    {'name': 'Anna', 'grade': 92, 'age': 20}
]
```

```python
# Sort by grade (descending) and then by age (ascending)
sorted_students = sorted(
    students,
    key=lambda x: (-x['grade'], x['age'])
)

print("\nSorted by grade (desc) and age (asc):")
for student in sorted_students:
    print(f"{student['name']}: Grade={student['grade']}, Age=
{student['age']}")
```

### 2. Ταξινόμηση με Πολλαπλά Κριτήρια

```python
# Product sorting with multiple criteria
products = [
    {'name': 'Laptop', 'price': 1000, 'rating': 4.5},
    {'name': 'Mouse', 'price': 25, 'rating': 4.5},
    {'name': 'Monitor', 'price': 300, 'rating': 4.8},
    {'name': 'Keyboard', 'price': 80, 'rating': 4.2}
]

# Sort by rating (desc) and then by price (asc)
sorted_products = sorted(
    products,
    key=lambda x: (-x['rating'], x['price'])
)

print("\nProducts sorted by rating (desc) and price (asc):")
for product in sorted_products:
    print(f"{product['name']}: Rating={product['rating']},
Price=${product['price']}")
```

### 3. Συνδυασμός με List Comprehension

```python
# Creating a sorted list of tuples with lambda
numbers = [1, 2, 3, 4, 5]
sorted_pairs = sorted(
    [(x, x**2) for x in numbers],
    key=lambda x: x[1],
    reverse=True
)
print(sorted_pairs)  # Output: [(5, 25), (4, 16), (3, 9), (2, 4), (1, 1)]
```

## Πρακτικές Εφαρμογές

### 1. Επεξεργασία Δεδομένων (Data Processing)

```python
# Process sales data
sales = [
    {'product': 'laptop', 'revenue': 2000, 'cost': 1600},
    {'product': 'mouse', 'revenue': 50, 'cost': 20},
    {'product': 'monitor', 'revenue': 600, 'cost': 400}
]

# Calculate profit for each product
sales_with_profit = list(map(
    lambda x: {**x, 'profit': x['revenue'] - x['cost']},
    sales
))
```

```python
# Sort by profit
sorted_by_profit = sorted(
    sales_with_profit,
    key=lambda x: x['profit'],
    reverse=True
)

print("\nProducts sorted by profit:")
for item in sorted_by_profit:
    print(f"{item['product']}: Profit=${item['profit']}")
```

## Συμβουλές και Βέλτιστες Πρακτικές

1. **Πότε να Χρησιμοποιείτε Lambda Functions:**
   - Για απλές, μίας γραμμής λειτουργίες
   - Όταν η συνάρτηση χρησιμοποιείται μόνο μία φορά
   - Ως παράμετρος σε συναρτήσεις όπως `sorted()`, `filter()`, `map()`
2. **Πότε να Αποφεύγετε Lambda Functions:**
   - Για πολύπλοκη λογική
   - Όταν χρειάζεστε πολλαπλές εκφράσεις
   - Όταν η συνάρτηση θα επαναχρησιμοποιηθεί
3. **Καλές Πρακτικές:**
   - Κρατήστε τις lambda συναρτήσεις απλές και ευανάγνωστες
   - Χρησιμοποιήστε περιγραφικά ονόματα μεταβλητών
   - Προτιμήστε κανονικές συναρτήσεις για πολύπλοκη λογική

In [ ]: