sandeepsuryaprasad / **python_tutorials** Private

Code Issues Pull requests Actions Projects Security Ins

master python_tutorials / 7_sorting / **_sorting.py** /
Jump to

Go to file ...

Sandeep Suryaprasad testing Latest commit a868eb3 on 29 Nov 2021 **History**

0 contributors

171 lines (126 sloc) 5.19 KB Raw Blame

```python
# SORTING Iterables

names = ['apple', 'google', 'yahoo', 'amazon', 'facebook', 'instagram', 'micr

prices = {'ACME': 45.23, 'AAPL': 612.78, 'IBM': 205.55, 'HPQ': 37.20, 'FB': 1

numbers = (1, 2, 6, 7, 10, 3, 4, 5)

word = "hello"

# sorted method returns a new list in sorted order. # Original list remains u
sorted_names = sorted(names)

# sorts the list in decending order
reverse_names = sorted(names, reverse=True)

# Sotring strings
word = "helloworld"
sortd_word = sorted(word)

# SORTING TUPLES
sorted_numbers = sorted(numbers)

# Sorting Dictionary
# Sorts the keys of the dictionary in ascending order
sorted_dict = sorted(prices)
# ============================================================================
# Custom Sorting
names = ['apple', 'google', 'yahoo', 'amazon', 'facebook', 'instagram', 'micr

# Sorting the list based on the number of characters of the list item
print(sorted(names, key=len))

# Sorting the list based on the last character of the list item
items = ['bv', 'aw', 'dt', 'cu']
s = sorted(items, key=lambda item: item[-1])

# Sorting based on temperatures
def get_temp(item):
    return item[-1]

temperatures = [('Bangalore', 25), ('Delhi', 35), ('Chennai', 37), ('Mumbai',

# using normal function
sorted(temperatures, key=get_temp)

# using lambda expression
sorted(temperatures, key=lambda item: item[-1])

# Sorting the list based on first item of each inner list
items = [[2, 7], [7, 3], [3, 8], [8, 7], [9, 7], [4, 9]]
sorted(items, key= lambda item: item[0])

# Sorting the list based on last item of each inner list
sorted(items, key= lambda item: item[-1])

# Sorting Dictionary based on values
my_dict = {'a': 4, 'b': 3, 'c': 2, 'd': 1}
print(sorted(my_dict.items(), key=lambda item: item[1]))

# Sorting Dictionary based on share price
prices = { 'ACME': 45.23, 'AAPL': 612.78, 'IBM': 205.55, 'HPQ': 37.20, 'FB':
s_prices = sorted(prices.items(), key=lambda d: d[-1])
min_p, *_, max_p = sorted(prices.items(), key=lambda d: d[-1])

print(min_p)
print(max_p)

# OR

min_price = min(s_prices, key=lambda item: item[-1])
max_price = max(s_prices, key=lambda item: item[-1])
```

```python

portfolio = [
                {'name': 'IBM', 'shares': 100, 'price': 91.1},
                {'name': 'AAPL', 'shares': 50, 'price': 543.22},
                {'name': 'FB', 'shares': 200, 'price': 21.09},
                {'name': 'HPQ', 'shares': 35, 'price': 31.75},
                {'name': 'YHOO', 'shares': 45, 'price': 16.35},
                {'name': 'ACME', 'shares': 75, 'price': 115.65}
            ]

def get_share_name(item):
    return item['name']

def get_no_shares(item):
    return item['shares']

def get_share_price(item):
    return item['price']

# Sorts based on share name
sorted(portfolio, key=get_share_name)
sorted(portfolio, key=lambda d: d.get('name'))

# Sorts based on number of shares
sorted(portfolio, key=get_no_shares)
sorted(portfolio, key=lambda d: d.get('shares'))

# Sorts based on number of price
sorted(portfolio, key=get_share_price)
print(sorted(portfolio, key=lambda d: d.get('price')))

students = [
    {"name": "john", "grade": "A", "age": 26},
    {"name": "jane", "grade": "B", "age": 28},
    {"name": "dave", "grade": "B", "age": 22}
]

# Sorting by age
by_age = sorted(students, key=lambda item: item['age'])
by_grade = sorted(students, key=lambda item: item['grade'])
by_name = sorted(students, key=lambda item: item['name'])

# Find the longest sub-string in the below string
sentence = "This is a Programming language and Programming is fun"
words = sentence.split()
```

```python
d = { word: len(word) for word in words}
longest_word = sorted(d.items(), key=lambda item: item[-1])

# Find the longest non-repeating sub-string in the below string
sentence = "This is a Programming language and Programming is fun"
words = sentence.split()
d = { word: len(word) for word in words if words.count(word) == 1}
longest_non_repeating_word = sorted(d.items(), key=lambda item: item[-1])

# Find the most repeated word in the below string
sentence = "hi hello hi hello world hi universe hi world hello world hi world
words = sentence.split()
d = {word: words.count(word)  for word in words }
most_repeated_word = sorted(d.items(), key=lambda item: item[-1])[-1]

# Find the most repeated character in the below string
sentence = 'hi hello hi hi hiiiiii'
d = {c: sentence.count(c) for c in sentence}
most_repeated_character = sorted(d.items(), key=lambda item: item[-1])


# Sorting a Custom Class
class Employee:
    def __init__(self, fname, lname, salary):
        self.fname = fname
        self.lname = lname
        self.salary = salary

emp1 = Employee('steve', 'jobs', 90000)
emp2 = Employee('bill', 'gates', 80000)
emp3 = Employee('joseph', 'trev', 70000)

li_emp = [emp1, emp2, emp3]

# Sorting Employee objects based on salary
s = sorted(li_emp, key=lambda emp: emp.salary)

class Student:
    def __init__(self, name, grade, age):
        self.name = name
        self.grade = grade
        self.age = age

    def __str__(self):
        return f"Student({self.name}, {self.grade}, {self.age})"
```

```python

s1 = Student('dave', 'A', 26)
s2 = Student('john', 'C', 28)
s3 = Student('jane', 'B', 24)

_students = [s1, s2, s3]

# Sort student objects by age
by_age = sorted(_students, key=lambda item: item.age)
```