Posortuj tablicę

w najbardziej leniwy sposób

In [ ]:
```
sorted([1, 5, 6, 10, 23, 67, 2, 8], reverse=True) # szybsza
sorted([1, 5, 6, 10, 23, 67, 2, 8])[::-1]         # leniwsza, mniej pisania
```

Out[ ]:
```
[67, 23, 10, 8, 6, 5, 2, 1]
```

Książka adersowa

Import potrzebnych bilbiotek, interesuje mnie `pprint` ułatwiający formatowanie kontenerów pythona i `typing` do silniejszego typowania

In [ ]:
```
import pprint
from typing import Union
```

Definiowanie wymaganych funkcji. Zaznaczam że:

- `add_contact_to` przyjmuje zarówno `dict` jak i `list` jako argumenty więc wywołania:

  ```
  add_contact_to(contact)
  add_contact_to([contact1, contact2, contact3])
  ```

- `add_contact_to` zawsze waliduje wszystkie kontakty, oczekując pól **first_name**, **last_name** i **contact**
- `search_contact_in` i `print_all_contacts` są napisane w jednej linijce

In [ ]:
```
def print_all_contacts(adress_book: list):
    pprint.pprint(adress_book)
def add_contact_to(address_book: list[dict], new_contact: Union[list, dict]):
    contact = [new_contact] if type(new_contact) is dict else new_contact

    def validate(contact: dict):
        if not isinstance(contact, dict):
            raise TypeError('Contact must be a dict')
        if 'first_name' not in contact:
            raise ValueError('Contact must have a first name')
        if 'last_name' not in contact:
            raise ValueError('Contact must have a last name')
        if 'email' not in contact:
            raise ValueError('Contact must have email')

    # Validate Contacts
    for c in contact:
        validate(c)

    # Add Contact
    address_book.extend(contact)

def search_contact_in(address_book: list, search_term: dict):
    # Search key in address book and return the contact
    return [contact for contact in address_book if all(contact[k] == v for k, v in
```

In [ ]:
```
address_book=[]

contact_1={"first_name": "Kazimierz", "last_name": "Kiełkowicz", "email":"kkielkow
contact_2={"first_name": "Maciej", "last_name": "Zlotorowicz", "email":"zlotymacie

print_all_contacts(address_book)

add_contact_to(address_book, [contact_1, contact_2])

print_all_contacts(address_book)
```

```
[]
[{'email': 'kkielkowicz@pk.edu.pl',
  'first_name': 'Kazimierz',
  'last_name': 'Kiełkowicz'},
 {'email': 'zlotymaciej@gmail.com',
  'first_name': 'Maciej',
  'last_name': 'Zlotorowicz'}]
```

In [ ]:
```
search_contact_in(address_book, {"first_name":"Kazimierz"})
```

Out[ ]:
```
[{'first_name': 'Kazimierz',
  'last_name': 'Kiełkowicz',
  'email': 'kkielkowicz@pk.edu.pl'}]
```

In [ ]:
```
search_contact_in(address_book, {"last_name":"Kazimierz"})
```

Out[ ]:
```
[]
```

In [ ]:
```
search_contact_in(address_book, {"email":"kkielkowicz@pk.edu.pl"})
```

Out[ ]:
```
[{'first_name': 'Kazimierz',
  'last_name': 'Kiełkowicz',
  'email': 'kkielkowicz@pk.edu.pl'}]
```

Klasa Rocket & Position

Definicja klasy `Position` przetrzymuje `x` i `y` i przeciąża kilka operatorów. Klasa Rocket dziedziczy po `Position` i dokłada swoje pola `name` i `fuel`. Funkcja move rusza statek i zmniejsza ilość paliwa.

In [ ]:
```
class Position:
    def __init__(self, x: int = 0, y: int = 0):
        self.x = x
        self.y = y
    def __add__(self, other):
        return Position(self.x + other.x, self.y + other.y)
    def __str__(self):
        return f'x: {self.x}, y: {self.y}'
    def __repr__(self):
        return f'Position(x={self.x}, y={self.y})'
```

In [ ]:
```
class Rocket(Position):
    def __init__(self, name: str, fuel: int = 10, x: int = 0, y: int = 0):
        super().__init__(x, y)
        self.name = name
        self.fuel = fuel
    def move(self, direction: Position):
        if self.fuel < 1:
            raise RuntimeError(f"Rocket '{self}' has no fuel 💥 Exploads 💥")
```

```
        self.fuel -= 1
        self += direction
    def __str__(self):
        return f'Rocket {self.name} at {super().__str__()}'
    def __repr__(self):
        return f'Rocket(name={self.name}, fuel={self.fuel}, x={self.x}, y={self.y}
```

In [ ]:

```
from random import randint

rockets = [Rocket(f'R{i}', randint(1, 10)) for i in range(1, 11)]
pprint.pprint(rockets)

while len(rockets) != 0:
    for rocket in rockets:
        try:
            rocket.move(Position(randint(-1, 1), randint(-1, 1)))
        except RuntimeError as err:
            print(err)
            rockets.remove(rocket)
```

```
[Rocket(name=R1, fuel=3, x=0, y=0),
 Rocket(name=R2, fuel=3, x=0, y=0),
 Rocket(name=R3, fuel=8, x=0, y=0),
 Rocket(name=R4, fuel=5, x=0, y=0),
 Rocket(name=R5, fuel=9, x=0, y=0),
 Rocket(name=R6, fuel=5, x=0, y=0),
 Rocket(name=R7, fuel=9, x=0, y=0),
 Rocket(name=R8, fuel=5, x=0, y=0),
 Rocket(name=R9, fuel=8, x=0, y=0),
 Rocket(name=R10, fuel=8, x=0, y=0)]
Rocket 🚀 'Rocket R1 at x: 0, y: 0' has no fuel 💥 Exploads 💥
Rocket 🚀 'Rocket R2 at x: 0, y: 0' has no fuel 💥 Exploads 💥
Rocket 🚀 'Rocket R4 at x: 0, y: 0' has no fuel 💥 Exploads 💥
Rocket 🚀 'Rocket R6 at x: 0, y: 0' has no fuel 💥 Exploads 💥
Rocket 🚀 'Rocket R8 at x: 0, y: 0' has no fuel 💥 Exploads 💥
Rocket 🚀 'Rocket R10 at x: 0, y: 0' has no fuel 💥 Exploads 💥
Rocket 🚀 'Rocket R3 at x: 0, y: 0' has no fuel 💥 Exploads 💥
Rocket 🚀 'Rocket R9 at x: 0, y: 0' has no fuel 💥 Exploads 💥
Rocket 🚀 'Rocket R7 at x: 0, y: 0' has no fuel 💥 Exploads 💥
Rocket 🚀 'Rocket R5 at x: 0, y: 0' has no fuel 💥 Exploads 💥
```