Лабораторная работа №3-4

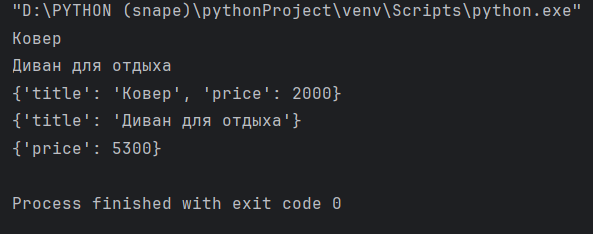
По дисциплине «Парадигмы и конструкции языков программирования»

ИБМ3-34Б

Новачлы Надежда

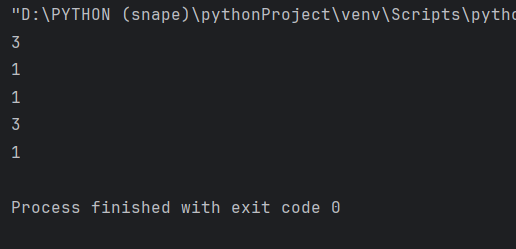
№1

def field(items, \*args):  
 assert len(args) > 0  
 for item in items:  
 if len(args) == 1:  
 # Если передан только один аргумент, выдаем значения полей  
 value = item.get(args[0])  
 if value is not None:  
 yield value  
 else:  
 # Если передано несколько аргументов, формируем словарь  
 result = {arg: item.get(arg) for arg in args}  
 # Удаляем ключи с None значениями  
 result = {k: v for k, v in result.items() if v is not None}  
 if result: # Если результат не пустой, выдаем его  
 yield result  
goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'price': None, 'color': 'black'},  
 {'title': None, 'price': 5300, 'color': 'blue'},  
]  
  
# Генератор для одного поля  
for title in field(goods, 'title'):  
 print(title) # Вывод: Ковер, Диван для отдыха  
  
# Генератор для нескольких полей  
for item in field(goods, 'title', 'price'):  
 print(item)



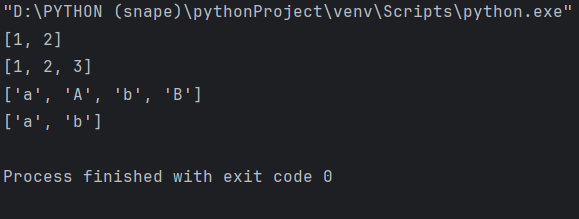
№2

import random  
  
def gen\_random(num\_count, begin, end):  
 for \_ in range(num\_count):  
 yield random.randint(begin, end)  
for number in gen\_random(5, 1, 3):  
 print(number)



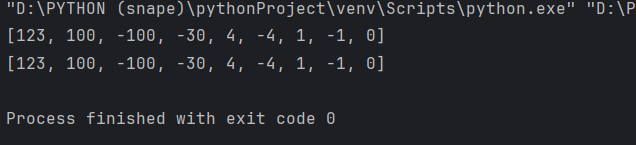
№3

class Unique:  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.items = items  
 self.ignore\_case = kwargs.get('ignore\_case', False)  
 self.seen = set()  
 self.iterator = iter(items)  
  
 def \_\_iter\_\_(self):  
 return self  
  
 def \_\_next\_\_(self):  
 for item in self.iterator:  
 # Определяем, как сравнивать элементы в зависимости от ignore\_case  
 key = item.lower() if self.ignore\_case and isinstance(item, str) else item  
 if key not in self.seen:  
 self.seen.add(key)  
 return item  
 raise StopIteration  
  
  
data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
unique1 = Unique(data1)  
print(list(unique1))   
  
data2 = (x for x in range(1, 4)) # Генератор  
unique2 = Unique(data2)  
print(list(unique2))   
  
data3 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
unique3 = Unique(data3)  
print(list(unique3))   
unique4 = Unique(data3, ignore\_case=True)  
print(list(unique4))



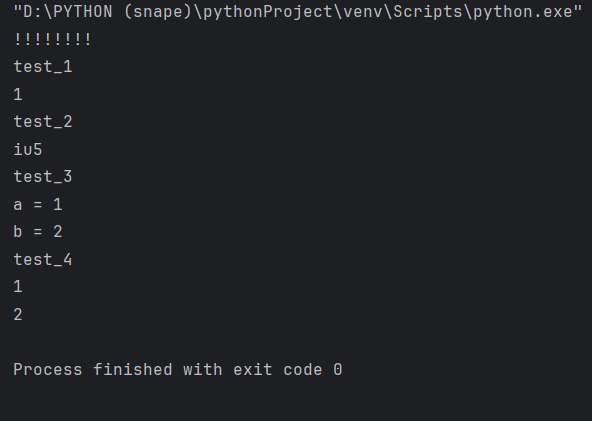
№4

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Без использования lambda-функции  
 result = sorted(data, key=abs, reverse=True)  
 print(result)  
  
 # С использованием lambda-функции  
 result\_with\_lambda = sorted(data, key=lambda x: abs(x), reverse=True)  
 print(result\_with\_lambda)



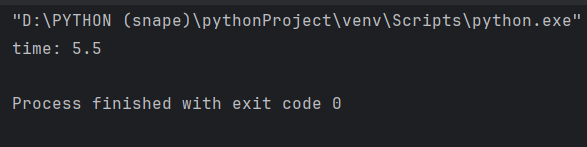
№5

def print\_result(func):  
 def wrapper(\*args, \*\*kwargs):  
 result = func(\*args, \*\*kwargs)  
 print(func.\_\_name\_\_) # Выводим имя функции  
  
 if isinstance(result, list):  
 for item in result:  
 print(item) # Выводим элементы списка в столбик  
 elif isinstance(result, dict):  
 for key, value in result.items():  
 print(f"{key} = {value}") # Выводим ключи и значения словаря в столбик  
 else:  
 print(result) # Для других типов просто выводим результат  
  
 return result # Возвращаем результат выполнения функции  
  
 return wrapper  
  
  
@print\_result  
def test\_1():  
 return 1  
  
  
@print\_result  
def test\_2():  
 return 'iu5'  
  
  
@print\_result  
def test\_3():  
 return {'a': 1, 'b': 2}  
  
  
@print\_result  
def test\_4():  
 return [1, 2]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print('!!!!!!!!')  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()



№6

import time  
  
class cm\_timer\_1:  
 def \_\_enter\_\_(self):  
 self.start\_time = time.time()  
 return self  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 end\_time = time.time()  
 elapsed\_time = end\_time - self.start\_time  
 print(f'time: {elapsed\_time:.1f}')  
  
# Пример использования  
if \_\_name\_\_ == "\_\_main\_\_":  
 from time import sleep  
  
 with cm\_timer\_1():  
 sleep(5.5)  
import time  
from contextlib import contextmanager  
  
@contextmanager  
def cm\_timer\_2():  
 start\_time = time.time()  
 yield  
 end\_time = time.time()  
 elapsed\_time = end\_time - start\_time  
 print(f'time: {elapsed\_time:.1f}')  
  
# Пример использования  
if \_\_name\_\_ == "\_\_main\_\_":  
 from time import sleep  
  
 with cm\_timer\_2():  
 sleep(5.5)



№7