УРОК 27. ВВЕДЕНИЕ В ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ НА PYTHON

ВИДЫ ПРОГРАММИРОВАНИЯ

Парадигма программирования представляет собой подход или методологию, которая определяет способ структурирования, организации и написания программного кода. В рамках каждой парадигмы программирования есть свои основные принципы, концепции и инструменты. Некоторые из популярных парадигм программирования включают императивное, декларативное, объектно-ориентированное, функциональное и логическое программирование.

- Императивное программирование описывает шаги и инструкции для выполнения конкретной задачи.
- Декларативное программирование фокусируется на описании желаемого результата, а не на определении шагов для его достижения.
- Объектно-ориентированное программирование ставит объекты в центр разработки и использует понятия классов, наследования и инкапсуляции.
- Функциональное программирование ориентировано на использование функций в качестве основной строительной единицы программы и ставит акцент на неизменяемость данных.
- Логическое программирование основано на формальной логике и использует правила и факты для вывода результатов.

Функциональное программирование подразумевает использование функций в качестве основной строительной единицы программы.

Основные принципы функционального программирования включают неизменяемость данных, отсутствие побочных эффектов и использование функций высшего порядка. Функциональное программирование часто используется для решения задач, где необходимо обработать данные, преобразовать их и получить результат, не изменяя исходные данные. Этот подход также полезен при работе с

параллельными и распределенными системами, а также при написании чистого и модульного кода.

Python предлагает ряд встроенных функций для работы с последовательностями, такими как списки, кортежи и строки. Некоторые из этих функций включают len(), который возвращает длину последовательности, sum(), который возвращает сумму элементов последовательности, min() и max(), которые возвращают наименьший и наибольший элементы соответственно.

```python
numbers = [1, 2, 3, 4, 5]
length = len(numbers) # 5
total = sum(numbers) # 15
minimum = min(numbers) # 1
maximum = max(numbers) # 5
```

ФУНКЦИИ ДЛЯ РАБОТЫ С ПОСЛЕДОВАТЕЛЬНОСТЯМИ

Однако, также существуют функции all() и any().

Функция all() возвращает True, если все элементы итерируемого объекта являются истинными, и False в противном случае.

Функция any() возвращает True, если хотя бы один элемент итерируемого объекта является истинным, и False в противном случае.

```python
numbers = [1, 2, 3, 4, 5]
all_true = all(num > 0 for num in numbers) # True
any_true = any(num < 0 for num in numbers) # False
```

ЗАДАНИЕ ДЛЯ ЗАКРЕПЛЕНИЯ

Что выведет следующий фрагмент кода:

```python
item_list = []
print (all(item_list))
```

```python
item_list = []
print (all(item_list))
```

```
True
```

ОТЛИЧИЯ SORTED И REVERSED В ФУНКЦИОНАЛЬНОМ ПРОГРАММИРОВАНИИ

Функция sorted() принимает итерируемый объект и возвращает новый список, содержащий отсортированные элементы исходного объекта.
Эта функция не изменяет исходный объект.

Функция reversed() принимает итерируемый объект и возвращает обратный итератор, который позволяет перебирать элементы в обратном порядке.
Эта функция не изменяет исходный объект и не создает новый список.

```python
numbers = [3, 1, 4, 2, 5]
sorted_numbers = sorted(numbers) # [1, 2, 3, 4, 5]
reversed_numbers = reversed(numbers) # <list_reverseiterator object at
0x7f4b930b3d00>
```

ЗАДАНИЕ ДЛЯ ЗАКРЕПЛЕНИЯ

Что будет выведено в результате выполнения фрагмента кода:

```python
numb = (22, 10, 5, 34, 29)
print(list(reversed(numb)))
```

```python
numb = (22, 10, 5, 34, 29)
print(list(reversed(numb)))

# Output:
# [29, 34, 5, 10, 22]
```

ВСТРОЕННЫЕ МЕТОДЫ ENUMERATE И ZIP

Функция enumerate() принимает итерируемый объект и возвращает итератор, который генерирует кортежи, состоящие из индекса элемента и самого элемента.

Это полезно, когда необходимо получить доступ к индексам элементов во время итерации.

```python
fruits = ['apple', 'banana', 'orange']
for index, fruit in enumerate(fruits):
  print(index, fruit)

# Output:
# 0 apple
# 1 banana
# 2 orange
```

Функция zip() принимает несколько итерируемых объектов и возвращает итератор, который генерирует кортежи, содержащие элементы из каждого итерируемого объекта на соответствующих позициях.

Это позволяет объединять элементы из разных последовательностей.

```python
numbers = [1, 2, 3]
letters = ['a', 'b', 'c']
zipped = zip(numbers, letters)
for num, letter in zipped:
  print(num, letter)

# Output:
# 1 a
# 2 b
# 3 c
```

ЗАДАНИЕ ДЛЯ ЗАКРЕПЛЕНИЯ

Что будет выведено в результате выполнения фрагмента кода:

Python

```
colour = ["Black", "Purple", "Brown", "Yellow", "Blue"]
print(list(enumerate(colour)))
```

Python

```
colour = ["Black", "Purple", "Brown", "Yellow", "Blue"]
print(list(enumerate(colour)))

# Output:
# [(0, 'Black'), (1, 'Purple'), (2, 'Brown'), (3, 'Yellow'), (4, 'Blue')]
```

ВВЕДЕНИЕ В MAP-FILTER-REDUCE

map, filter и reduce - это функции высшего порядка, которые широко используются в функциональном программировании.

map применяет функцию к каждому элементу итерируемого объекта и возвращает итератор с преобразованными значениями.

filter применяет функцию-предикат к каждому элементу итерируемого объекта и возвращает итератор, содержащий только элементы, для которых предикат возвращает True.

reduce применяет функцию двух аргументов к элементам итерируемого объекта, последовательно сводя их к одному значению.

```python
numbers = [1, 2, 3, 4, 5]
squared = map(lambda x: x ** 2, numbers) # [1, 4, 9, 16, 25]

even = filter(lambda x: x % 2 == 0, numbers) # [2, 4]

from functools import reduce
product = reduce(lambda x, y: x * y, numbers) # 120
```

ПРАКТИЧЕСКАЯ РАБОТА

Считать данные из файла anna-karenina.txt, очистить их, оставить только слова длиной более десяти символов.

ПОЛЕЗНЫЕ МАТЕРИАЛЫ

1. Введение в функциональное программирование на Python
2. Python/Функциональное программирование на Python