ГЕНЕРАТОР УСЛОВИЙ ЗАДАЧ (№10)

Команда:
9303 Гугунов Сергей
1381 Возмитель Влас
1381 Тарасов Константин
1381 Харитонов Никита
1381 Манучарова Ангелина
1384 Степаненко Денис

План на текущую итерацию

- Подготовить первую версию:
  - Реализовать часть подтипов задач
  - Создать главный файл, связывающий все подтипы
  - Создать README.md, где видны все материалы
  - Описать сценарий использования программы
  - Написать инструкцию по настройке и dockerfile | docker-compose

Процесс реализации задач

Были проведены регулярные созвоны команды, составлены протоколы встреч, определены задачи и обязанности:

- Реализовать тип Синтаксис, подтип - изменение количества повторений в циклах
- Реализовать тип Затирание кода, подтип - затирание части строки
- Реализовать тип Синтаксис, подтип - изменение типа переменных
- Реализовать тип Логика, подтип - замена знаков в условиях
- Реализовать тип Логика, подтип - ошибки в возвращаемом аргументе
- Реализовать тип Синтаксис, подтип - ошибки, подсвечиваемые редактором кода
- Реализовать main файл
- Написать сценарий использования
- Написать и оформить README.md
- Docker

Список реализованных подтипов задач

- Изменение типа переменных
- Изменение количества итераций в цикле for
- Изменение условий в коде
- Полное или частичное затирание определенных вещей в коде
- Добавление ошибок, подсвечиваемых редактором кода
- Изменение порядка подаваемых в функцию аргументов (или их затирание)
- Изменение возвращаемого из функции аргумента
- Добавление ошибки, связанной с областью видимости

Изменение типа переменных

Было:

```
def search(n, arr):
    k = 0
    m = 9
    d = 8
    return k

none_ex = None
bool_ex = True
str_ex = 'example'
float_ex = 9.9
```

```
list1 = [1, 2 , 3]
list2 = [1, 2, 3]
list3 = [1.0, 2.0, 3.0]
list4 = [True, False, True, None]
```

Стало:

```
def search(n, arr):
    k = 0
    m = '9'
    d = 8
    return k

none_ex = None
bool_ex = None
str_ex = 'example'
float_ex = 9.9
```

```
list1 = ['1', '2', '3']
list2 = [1.0, 2.0, 3.0]
list3 = ['1.0', '2.0', '3.0']
list4 = ['True', 'False', 'True', 'None']
```

Изменение типа переменных –

- *Изменяется тип на другой логически возможный*
- *В списках: int->int/float/str; float->float/str; other->other/str*

Изменение количества итераций в цикле for

Было:

```
def find_max():
    for i in range(10):
        if arr[i] > max:
            max = arr[i]
    return max
```

```
for i in range(0, 6):
        print('example')
return 1
```

Стало:

```
def find_max():
    for i in range(23):
        if arr[i] > max:
            max = arr[i]
    return max
```

```
for i in range(0, 11):
        print('example')
return 1
```

Изменение количества итераций –

- *В цикле for()*

Изменение условий в коде

Было:

```
while low <= high:
    middle = (low + high) // 2
    if arr[middle] == elem:
        return middle
    elif arr[middle] > elem:
        high = middle - 1
    else:
        low = middle + 1
```

Стало:

```
while low == high:
    middle = (low + high) // 2
    if arr[middle] == elem:
        return middle
    elif arr[middle] == elem:
        high = middle - 1
    else:
        low = middle + 1
```

```
if cur and tmp:
        return True
```

```
if cur or tmp:
        return True
```

Изменение условий в коде –

- *Замена знаков сравнения на любой другой*
- *Замена and на or и наоборот*

Полное или частичное затирание в коде

Было:

```
def binary_search(arr, elem):
    low = 0
    high = len() - 1
```

```
if arr[middle] == elem:
    return middle
```

```
list1 = ["12.0", 24.0, 32, 39.0, 45.0,
50.0, None]
list2 = [12, 24, 32, 39, 45, 50, 54]
list3 = [1.0, 2.0, 3.0]
list4 = [True, False, True, None]
```

```
def search(n, arr):
    k = 0
    m = 9
    d = 8
    return
```

Стало:

```
def binary_search(arr, elem):
    ___ = 0
    high = len() - 1
```

```
if arr[middle] == elem:
    ____________
```

```
list1 = ['12.0', 24.0, 32, 39.0, 45.0,
None]
list2 = [12, 24, 32, 39, 45, 50]
list3 = [2.0, 3.0]
list4 = [False, True]
```

```
def search(arr):
    k = 0
    m = 9
    d = 8
    return
```

Полное или частичное затирание в коде –

- *Удаление параметров-переменных, которые передаются в функцию*
- *Удаление некоторых элементов списка*
- *Удаление имени переменной*
- *Частичное или полное удаление строки return*

Ошибки, подсвечиваемых редактором кода

Было:

```
def binary_search(arr, elem):
    low = 0
    high = len(arr) - 1
    print(high)
    ...
```

Стало:

```
def binary_search(arr, elem):
    low = 0
    high = len(arr) - 1
    print(high)
    ...
```

Подтип задач, подсвечиваемых редактором кода - замена букв латинского алфавита в названиях функций на буквы кириллицы. В данном примере a(EN) -> а(RU) в имени функции.

Порядок подаваемых в функцию аргументов

Было:

```
result = binary_search(list1, n)
search(n, list1)

if result != -1:
    print('Element is present at index',
str(result))
else:
    print('Element is not present in arr')
```

Стало:

```
result = binary_search(n, list1)
search(list1, n)

if result != -1:
    print('Element is present at index',
str(result))
else:
    print('Element is not present in arr')
```

```
result = binary_search(list1, n)
search(n, list1)
```

```
result = binary_search(____)
search(____)
```

Порядок подаваемых в функцию аргументов -

- *Изменение порядка аргументов*
- *Затирание аргументов*

Изменение возвращаемого аргумента

Было:

```
while low <= high:
    middle = (low + high) // 2
    if arr[middle] == elem:
        return middle
    elif arr[middle] > elem:
        high = middle - 1
    else:
        low = middle + 1
```

Стало:

```
while low <= high:
    middle = (low + high) // 2
    if arr[middle] == elem:
        return low
    elif arr[middle] > elem:
        high = middle - 1
    else:
        low = middle + 1
```

Изменение возвращаемого аргумента –

- *Происходит поиск всех переменных, объявленных в функциях. Затем замена возвращаемого аргумента другим из списка переменных*

Ошибки, связанной с областью видимости

Было:

```
def binary_search(arr, elem):
    low = 0
    high = len(arr) - 1
    print(high)

...

result = binary_search(list1, n)
search(n, list1)
if result != -1:
    print(some txt ', str(result))
else:
    print('some txt')
```

Стало:

```
def binary_search(arr, elem):
    low = 0
    high = len(arr) - 1
    print(high)

...

result = binary_search(list1, n)
search(n, list1)
if result != -1:
    print(some txt ', str(result))
else:
    print('some txt')
high = 42 #новая строка
print(high) #новая строка
```

Ошибки, связанной с областью видимости –

- *Для использования этого функционала в созданной функции должна быть объявлена переменная и вывод её с помощью print. В функции main создается переменная с таким же именем, как и у переменной из функции. Затем эта переменная выводится. Соответственно, если вызвать функцию, то значение этой переменной будет иным.*

Создание главного файла, readme, сценария использования, docker

Также:

- Был создан главный файл main.py, который соединяет в себя все подтипы задач
- Создан и оформлен README.md, где изложены все материалы проекта
- Написан сценарий использования для запуска программы
- Обернули текущую версию в docker

Сценарий использования

После запуска файла main.py в консоли будет предложено ввести одну из цифр (от 1 до 6), каждая из которых решает разные задачи:

- 1 - Изменить названия функций
- 2 - Удалить строки
- 3 - Изменить аргументы передаваемые в функцию/возвращаемые функцией и область видимости
- 4 - Заменить тип переменных
- 5 - Изменить количество повторений в цикле for
- 6 - Поменять операции сравнения и булевы операции

Также предусмотрены команды help, которая выведет справку по каждой из команд

Программа работает циклически и можно выполнять разные действия без перезапуска программы. Код, который будет использоваться должен быть на языке python и находиться в файле code.txt в той же директории, что и main.py. Результат работы программы будет в файле output.txt.

Сценарий использования

Если запуск программы не предусматривает загрузку необходимых компонентов, то программу следует запускать через docker. Для его запуска необходимо выполнить 2 действия:

- 1 - создать образ с помощью команды docker build -t <любое имя программы> .

  например: docker build -t prog .

  Примечание: точка в конце обязательна

- 2 - создать контейнер и запустить программу командой docker run -it <название программы (образа)>

  например: docker run -it prog

План на следующую итерацию

- Дописать функционал программы (версия 2)
- Реализовать базовые тесты (интеграционные, функциональные)