Python-13
Интерпретатор ЯП.

Что такое интерпретатор ЯП?

Интерпретатор языка программирования (ЯП) - это программное средство, которое выполняет код, написанный на определенном языке программирования. Вместо того чтобы компилировать весь исходный код в машинный код заранее (как делает компилятор), интерпретатор работает пошагово, выполняя инструкции программы одну за другой в реальном времени. Это позволяет программистам писать и тестировать код пошагово, без необходимости явной компиляции перед каждым выполнением.

AlaVis v1.0

AlaVis- это язык программирования построенный на базе популярного языка Python. Предназначен для решения простых задач, для работы с внутренностями компьютера. Синтаксис языка очень прост в исполнении. Интерпретатор считывает построчно код и выполняет все последовательно (*сверху вниз по файлу* **.av**).

1.2 Операторы ввода/вывода.

Основными операторами языка являются операторы ввода и вывода какие либо данных:

- **<<#** оператор ввода данных
- **#>>** оператор вывода данных

```
#>> Hello, world!
```

AlaVis. Переменные.

1.3 Переменные и типизация данных. Преобразование данных.
Переменная - это как ярлык или контейнер, который хранит определенное значение или данные в программе.

Данные в **AlaVis**могут быть следующими типами:

- **int** целочисленные данные
- **float** вещественные данные
- **string** - строковые данные
- **bool** - булевые данные (*True или False*)

Пример создания переменной (*тип данных определяется автоматически как string*):

```
<<# num1 123
```

В данном примере мы создали переменную с именем **num1** и строковым значением по-умолчанию 123.

AlaVis. Явная типизация.

Явное указание типа данных при создании переменной:

```
<<# num1 int 123
```

В данном примере мы создали переменную с именем **num1** и целочисленным значением 123.

Метод **typed** - используется для проверки типа данных у заданной переменной.

Пример:

```
typed num1
```

Вывод:

```
typed num1 - int
```

Метод **formed** - используется для изменения (обновления) типа данных у заданной переменной. (например поменять string в float)

Пример:

```
formed num1 int
```

Вывод: нету

Математические операции с переменными.

Основными операциями над переменными являются:
+ сложение
- вычитание
***** умножение
/ деление
% остаток от деления

Все математические операции записываются в квадратных скобках […]
Пример (*разность двух чисел типа int*):

```
<<# num1 int 123
<<# num2 int 12
<<# num3 int [ num1 - num2 ]
#>> num3
```

Сравнительный метод.

Сравнительный метод comp - используется для сравнения двух переменных (на равенство, больше или меньше и др).

Основные конструкции:
comp_equal - истина если две переменные равны.
comp_not_equal - истина если две переменные не равны.
comp_less - истина если переменная1 < переменной2.
comp_more - истина если переменная1 > переменной2.
comp_eless - истина если переменная1 <= переменной2.
comp_emore - истина если переменная1 >= переменной2.

конструкция завершение comp **::** - означает конец блока сравнительных операций.

(*иными словами - блок сравнительных операций продолжается до тех пор, пока не встретится специальный символ ::, который означает конец блока.*)

```
<<# num1 int 123
<<# num2 int 12
<<# num3 int [ num1 - num2 ]
#>> num3

comp_equal num1 num2
typed num3
::
```

Цикличный метод.

Цикл в программировании - это конструкция, позволяющая выполнять определенный блок кода несколько раз. Это полезно, когда вам нужно повторять одни и те же действия несколько раз, например, для обработки множества данных или выполнения однотипных операций.

В **AlaVis** есть один единственный (*и простой*) цикл **loop**. (*работает примерно также как типичный while цикл*)

Дополнительные методы.

max - максимальное значение из..

```
<<# n1 float 5
<<# n2 float 0
<<# n3 float 1
<<# n float 0
max n n1 n2 n3
#>> n
```

sqrt - квадратный корень от числа (auto - float)
пример - корень из числа 4

```
sqrt n 4
#>> n
```

pow - возведение числа в степень (auto - float)
пример - число 3 в степень 2. т.е 3^2

```
pow n 3 2
#>> n
```

abs - модуль от числа (auto - float)

```
abs n -5
#>> n
```

gcd - наибольший общий делитель (НОД) (auto - int)

```
gcd n n1 n2 n3
#>> n
```

lcm - наименьшее общее кратное (НОК) (auto - int)

```
lcm n n1 n2 n3
#>> n
```

del - удалить переменную

```
del n1
```

Модули

2.1 Модули. Подключение модулей.
Модуль **MATH** - используется для расширенных математических операций.
Модуль **FILE** - используется для работы с файлами и директориями.
Модуль **SYSTEM** - используется для управления ОС.
Модуль **MEMORY** - используется для управления памятью компьютера.

Подключение модуля (*примеры*):

1. `/connect/ ? FILE`
2. `/connect/ ? MATH`

2.2 Модуль MATH.
Основные методы модуля:
sin - синус угла
cos - косинус угла
tan - тангенс угла
ctg - котангенс угла
arctan - арктангенс угла
arcctg - арккотангенс угла
pi - полная запись числа ПИ
integral - вычисление интеграла от выражения по dx
double_integral - двойной интеграл
factorial - факториал числа
log - логарифм
exp - экспонента
hypot - вычисление гипотенузы (теорема Пифагора)
round - округление числа

main.py part1

```
import os
import math
from datetime import datetime
from file import *
from maths import *
from sys import *
from mem import *

class Interpreter:
    def __init__(self) -> None:
        self.variables = {}
        self.back_res = None
        self.break_loop = None
        self.file_module = False
        self.math_module = False
        self.sys_module = False
        self.mem_module = False
        self.file_info = None
        self.file_status = False
        self.file_name = ""

    def execute(self, code):
        try:
            lines = code.split('\n')
            for line in lines:
                self.execute_line(line)
        except Exception as err:
            print(err)
```

main.py part2

```
def execute_line(self, line):
        tokens = line.split()
        if tokens:
            command = tokens[0]
            if command == "<<#":
                if (self.back_res == True or self.back_res == None) and (self.break_loop == None or self.break_loop == True):
                    typed = tokens[2]
                    variable_name = tokens[1]
                    if typed != "int" and typed != "string" and typed != "float" and typed != "bool":
                        self.variables[variable_name] = tokens[2]
                    else:
                        if tokens[3] == "[":
                            operand1 = None
                            operand2 = None

                            if typed == "int":
                                operand1 = int(self.variables[tokens[4]])
                                operand2 = int(self.variables[tokens[6]])
....
```

main.py part3

```
        elif typed == "string":
            operand1 = self.variables.get(tokens[4], str(tokens[4]))
            operand2 = self.variables.get(tokens[5], str(tokens[5]))
        elif typed == "float":
            operand1 = self.variables.get(tokens[4], float(tokens[4]))
            operand2 = self.variables.get(tokens[5], float(tokens[5]))
        elif typed == "bool":
            operand1 = self.variables.get(tokens[4], bool(tokens[4]))
            operand2 = self.variables.get(tokens[5], bool(tokens[5]))

        result = 'NaN'
        if tokens[5] == '+':
            result = operand1 + operand2
        elif tokens[5] == '-':
            result = operand1 - operand2
        elif tokens[5] == '*':
            result = operand1 * operand2
        elif tokens[5] == '/':
            result = operand1 / operand2
        self.variables[variable_name] = result
    else:
        if typed == "int":
            self.variables[variable_name] = int(tokens[3])
        elif typed == "string":
            self.variables[variable_name] = str(tokens[3])
        elif typed == "float":
            self.variables[variable_name] = float(tokens[3])
        elif typed == "bool":
            self.variables[variable_name] = bool(tokens[3])
```

....

main.py part4

```
elif command ==  ":.":
            self.break_loop =  None
            elif command ==  "fi":
                if self.file_module:
                    with open(f"{self.file_name}", "r") as file:
                        self.file_info =  file.readline()
                    print(self.file_info)
                else:
                    print("Error! The File module is not connected or connected with an error!   ")
                    exit()
            elif command ==  "savefi":
                if self.file_module:
                    self.variables[tokens[ 1]] = self.file_info
                else:
                    print("Error! The File module is not connected or connected with an error!   ")
                    exit()
            elif command  in ["open", "read", "write", "awrite"]:
                if self.file_module:
                    if command ==  "open":
                        self.file_status =  True
                    if self.file_status:
                        lst = list()
                        for i in range(len(tokens)):
                            try:
                                lst.append(self.variables[tokens[ i]])
                            except:
                                lst.append(tokens[ i])
                        try:
                            if self.file_name ==  "" or self.file_name ==  None:
                                self.file_name = tokens[ 1]
                        except:
                            if self.file_name ==  "" or self.file_name ==  None:
                                self.file_name =  ""
                        file = File(command,  lst, str(self.file_name))
                        temp = file.run()
                        if temp != None:
                            self.file_info =  temp
                    else:
                        print("Error! You forgot to open the stream files!  ")
                        exit()
                else:
                    print("Error! The File module is not connected or connected with an error!   ")
```