Sython文法规格以及解释器说明文档

小组成员

- ZY1906208 王帅
- SY1906418 李坤浩
- BY1906045 孙舟涛
- ZY1906804 王雅卉
- SY1906422 许京爽

介绍

Sython取名来源于Simple Python，即为简单的Python语言，它是一门无类型、面向对象的脚本语言，在基本语法层面上，较为类似C语言。同时，它是解释型语言，即Sython解释器将源代码转换为语法树，然后再由解释器分析语法树并执行。

本小组从C语言的基本文法入手，通过自己对Python语言语法规则的理解以及一些其他资料的学习，以C文法为基础，设计了Sython文法，既支持面向对象的抽象编程也支持面向过程的函数编程。通过对lex&yacc的学习，本小组为Sython设计制作了词法语法分析器，并部分完成了Sython的解释器。

设计初衷

目前，Python的使用热度逐步上升，其特点是“优雅”、“明确”、“简单”，其最终的代码不但看起来简单美观，并且功能同样十分强大。初学者学Python，相比于学习C\C++等语言简单了很多。然而，正是由于其简单性，我们在学习Python的时候，很多时候都是蜻蜓点水，浅尝辄止。同时，Python语言也有一些让人又爱又恨的特性，例如不需

要花括号，只需要缩进，不需要分号，只需要换行。这样确实使得代码看起来简洁、美观，但是有时候一个小空格造成的小bug却困扰人很久。

通过对python语言和c/c++语言的各自特点的分析，考虑到无类型脚本语言现在的热度，以及我们科研项目的常用语言特点，我们试图从c语言出发，摒弃其复杂的操作。并且c语言操作复杂，对于新手不太友好，相比于Python，开发周期慢。因此，我们试图开发一种无类型、可扩充的脚本式语言。

本小组为加深对Python语言的了解以及深入学习程序语言的设计，从C语言出发，按照我们的设计，将其扩充为一种类似Python语言的脚本语言，Sython。相比于c/c++语言,Sython摒弃了复杂的指针，并对语法进行了调整，从而更简单易学，上手快，同时也具有可扩展性和可嵌入型。我们在Sython中引入了类，从而Sython既支持面向对象的抽象编程也支持面向过程的函数编程。相比与Python，Sython当然比不了，但是我们按照自己意图定制了一门语言，并拥有大部分Python的特性，这已经达到了我们的意图。

我们为Sython设计了解释器，在一步步的实现中，也加深了自己对于编译原理以及脚本语言的运行方式的理解。

Sython文法介绍

功能

目前，Sython支持的功能如下：

1. 变量

   支持局部变量、全局变量定义、支持变量引用、变量赋值

2. 基本数据类型（不需要声明）

   int、double、字符串、bool型

3. 运算

   数值：+、-、*、/、%、+=、-=、 *=、/=

   逻辑：<、>、!=、||、&&、|、&

方法调用：.

索引：[]

4. 控制结构

   支持if-else选择语句

   支持while循环

   支持for循环

   支持break退出循环

   支持continue跳出本次循环，开始下一轮循环

   支持return返回

5. 函数

   使用function实现函数定义

6. 类

   包括类定义和类实例

   支持类继承

   支持类成员

7. 注释

   使用#作为注释开始符号，换行自动结束

8. 异常处理

   支持try、catch语句

9. 包导入

   支持import关键字导入其他包

关键字

- function
- if
- else
- elsif

- while
- for
- return
- break
- continue
- null
- true
- false
- closure
- try
- catch
- global
- finally
- throw
- final
- class
- extend
- static

文法终结符定义

终结符

string	终结符	string	终结符
"("	LP	"<"	LT
")"	RP	"<="	LE
"["	LB	"+="	ADD_ASSIGN_T
"]"	RB	"-="	SUB_ASSIGN_T
"{"	LC	"*="	MUL_ASSIGN_T
"}"	RC	"/="	DIV_ASSIGN_T
";"	SEMICOLON	"%="	MOD_ASSIGN_T
","	COMMA	"!"	EXCLAMATION

string	终结符	string	终结符
"&&"	LOGICAL_AND	"~"	~
"||"	LOGICAL_OR	"+"	ADD
"="	ASSIGN_T	"++"	INCREMENT
"=="	EQ	"--"	DECREMENT
"!="	NE	"-"	SUB
">="	GE	"*"	MUL
"/"	DIV	"%"	MOD
"."	DOT	">"	GT

EBNF产生式

在本文法中，除""号之内包含的符号为终结符外， IDENTIFIER 、 INT_LITERAL 、 DOUBLE 、STRING 、TRUE 、FALSE 、 NULL也是终结符。为与yacc内的BNF尽量保持一致，非终结符没有使用大写符号。

非终结符	产生式
program	::= {translation_unit}
translation_unit	:: = {definition_or_statement} definition_or_statement
definition_or_statement	:: =function_definition| statement| def_class| def_calss
def_class	:: ="class" IDENTIFIER class_extend "{" field_def methods_def "}"
class_extend	:: = {"extend" IDENTIFIER}
field_def	:: = ["static"] instant_filed
instant_filed	:: = expression ";"
methods_def	:: = method_or_gs

非终结符	产生式
method_or_gs	:: ={method_def | getter_def | setter_def}
method_def	:: = ["static"] IDENTIFIER "(" [parameter_list] ")" block
getter_def	:: = ["static"] IDENTIFIER block
setter_def	:: = ["static"] "IDENTIFIER" "=" "(" block ")" block
function_definition	:: = "function" IDENTIFIER "(" [parameter_list] ")" block
parameter_list	:: = [parameter_list ","] IDENTIFIER
argument_list	:: = [argument_list ","] assignment_expression
expression	:: = [expression ","] assignment_expression
assignment_expression	::={["FINAL"] postfix_expression "=" | "+=" | "-=" | "*=" | "/=" | "%="} logical_or_expression
logical_or_expression	::= {lgical_and_expression "||" } logical_and_expression
logical_and_expression	::={equality_expression "&&" } equality_expression
equality_expression	::={relation_expression "==" | "!=" }relation_expression
relation_expression	::={additive_expression ">" | ">=" | "<="} additive_expression
additive_expression	::= {multicative_expression "+" | "-" } multiplicative_expression

非终结符	产生式
multiplicative_expression	::={unary_expression "*" | "/" | "%"} unary_expression
unary_expression	::={"-" | "!" }postfix_expression
postfix_expression	:: = {postfix_expression ("[" expression "]" | ("." IDENTIFIER) | ("(" [argument_list] ")") | "++" | "--"} primary_expression
primary_expression	::= "[" expression "]" | IDENTIFIER | INT_LITERAL | DOUBLE | STRING | TRUE | FALSE | NULL | array_literal | closure_defination
array_literal	::= "{" expression_list [","] "}"
closure_defination	::= "closure" [IDENTIFIER] "(" [parameter_list] ")" block
expression_list	::= assignment_expression { "," assignment_expression}
statement	expression ";" | global_statement | if_statement | while_statement | for_statement | foreach_statement | return_statement | break_statement | continue_statement | try_statement | throw_statement
global_statement	::= "global" identifier_list
identifier_list	::= IDENTIFIER{ "," IDENTIFIER}
if_statement	:: ="if" "(" expression ")" block [elsif_list] ["else" block]

非终结符	产生式
elsif_list	::= {elsif_list} elsif
elsif	::= "elsif" "(" expression ")" block
while_statement	::= "while" "(" expression ")" block
for_statement	::= "for" "(" [expression] ";" [expression] ";" [expression] ")" block
foreach_statement	::= "foreach" "(" IDENTIFIER ":" expression ")" block
return_statement	::= "return " [expression] ";"
break_statement	:: = "break" ";"
continue_statement	:: = "continue" ";"
try_statement	::= "try" block ["catch" "(" IDENTIFIER ")" block] ["finally" block]
throw_statement	::= "throw" expression ";"
block	::= "{" [statement_list] "}"

示范代码

代码说明

如需要测试自己的代码文法正确性，请使用语法分析器（见后说明）进行检测，有输出的可执行代码见运行说明一节

- 代码1和代码2是符合Sython文法的实例代码，语法分析器可以正确识别语法。

- 代码7/8/9是目前完成的解释器可以执行的代码，使用Ubuntu解释器或者Windows下的解释器均会有输出

代码1

```
class EM{
    q=5;
    create_point(x, y) {

     return this;
    }
    static test_static_method(){
    }
}

print("hhh");
```

预期结果，输出全部代码并输出no error!

```
(base) ricard@ubuntu:~/文档/研一/程序设计原理/最终大作业/interpreter/语法分析_src$ ./Sython 1.txt
class EM{
        q=5;
        create_point(x, y) {

         return this;
        }
        static test_static_method(){
        }
}

print("hhh");

>>>>>>>>>>no error!<<<<<<<<<<<<
```

代码2

```
#########################################################
#####
#test fuction defination
import nunmpy;
global a,b,c;
function show(){
    for(i=0;i<3;i=i+1){
        print(i);
    }
}
function show2(){
```

```
    for(i=0;i<4;i=i+1){
        print(i);
    }
}
#test funtion call
show();
show2();
import animal;
#test class defination
class Dog{
    name;
    #static str="wolf";
    #test method def
    new(name1){
        name=name1;
    }
    call(){
        print(str);
    }
    eat(food){

    }


}
#test class method call
dog1 = Dog.new("dog1");
dog1.eat("meat");
#test calculation
a=(2+5)*2/2-1;
a++;
a--;
b={1,2,3};
c={a,b};
p = create_point(10, 20);
#test try catch
try {
    a=1/0;
```

```
} catch (ex) {
    desc_exception(ex);
}

if(a==6){
    print("== successful");
}else{
    print("== error");
}
if(a<=6){
    print("<= successful");
}else{
    print("<= error");
}
if(a>=6){
    print(">= successful");
}else{
    print(">= error");
}
# test elsif
if(a>=6){
    print(">= successful");
}elsif(a==4){
    print("a == 4");
}elsif(a==3){
    print("a == 3");
}
else{
    print(">= error");
}
#test for while loop
for(i=0;i<10;i++){
    while(i<5){
        print(i);
    }
    if(i == 1){
        continue;
    }
```

```
    elsif(i == 9){
        break;
    }

}
```

语法分析器输出如下（输出过长，只截取了一部分，）：

```
}
dog1 = Dog.new("dog1");
dog1.eat("meat");
a=(2+5)*2/2-1;
a++;
a--;
b={1,2,3};
c={a,b};
p = create_point(10, 20);
try {
    a=1/0;
} catch (ex) {
    desc_exception(ex);
}

if(a==6){
        print("== successful");
}else{
        print("== error");
}
if(a<=6){
        print("<= successful");
}else{
        print("<= error");
}
if(a>=6){
        print(">= successful");
}else{
        print(">= error");
}
if(a>=6){
        print(">= successful");
}elsif(a==4){
        print("a == 4");
}elsif(a==3){
        print("a == 3");
}
else{
        print(">= error");
}
for(i=0;i<10;i++){
        while(i<5){
                print(i);
        }
        if(i == 1){
                continue;
        }
        elsif(i == 9){
                break;
        }

}

>>>>>>>>>>no error!<<<<<<<<<<<<
```

代码3

```
class Person{
```

```
    name, age, sex, ID, address;
    new( _name, _age, _sex, _ID){
        name = _name;
        age = _age;
        sex = _sex;
        ID = _ID;
    }
    setAddress(_address){
        address = _address;
    }
    getUp(){
        print(name);
        print(": get up.\n");
    }
    haveBreakfast(food){
        print(name);
        print(": have breakfast of ");
        print(food);
        print(".\n");
    }
    takeRest(){
        print(name);
        print(": take a rest.\n");
    }
    celebrateBirthday(){
        age += 1;
        print(name);
        print(": celebrate a birthday.\n");
        return age;
    }
    compareAge(p){
        if(this.age==p.age){
            print("The same age.");
        }
        elsif(this.age>p.age){
            print(this.name);
            print(" is older than ");
            print(p.name);
```

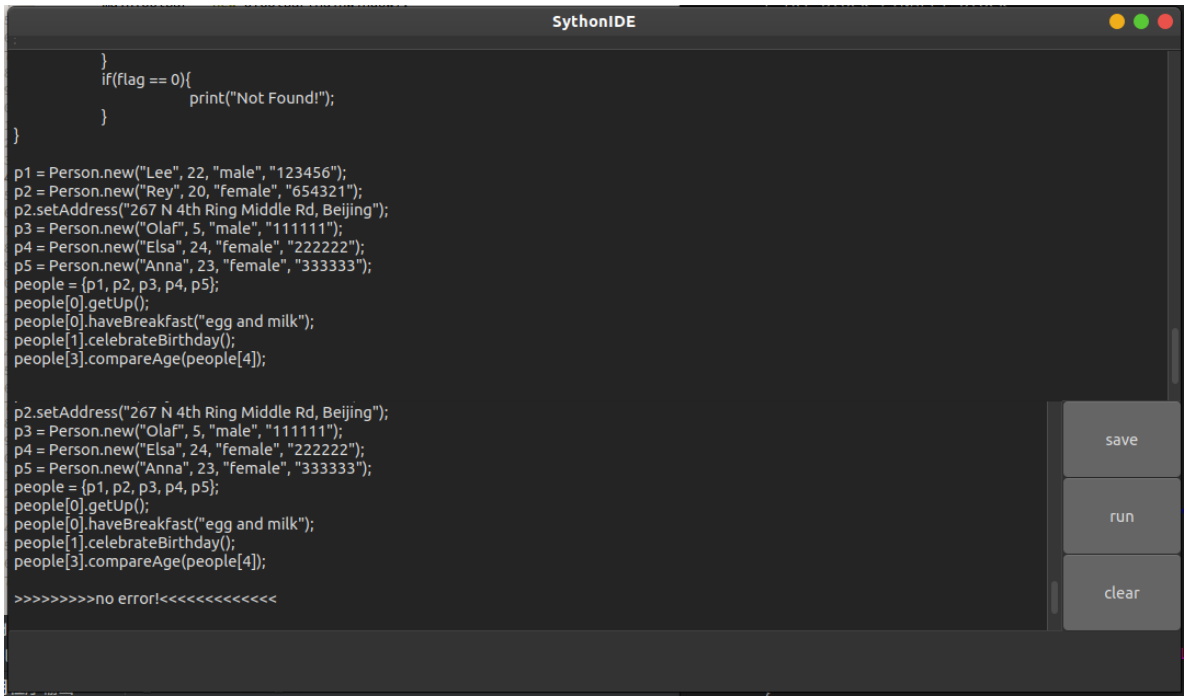
```
        }
        else{
            print(this.name);
            print(" is younger than ");
            print(p.name);
        }
    }
}

function getLength(arr){
    size = 0;
    foreach(num:arr){
        size = size +1;
    }
    return size;
}
function getSum(arr){
    res = 0;
    foreach(num:arr){
        res += num;
    }
    return res;
}

function getMean(arr){
    return getSum(arr) / getLength(arr);
}

function findPerson(name, arr){
    i = 0;
    flag = 0;
    for( ;i < getLength(arr); i++){
        if(arr[i].name == name){
            flag = 1;
            return arr[i];
        }
    }
    if(flag == 0){
```

```
            print("Not Found!");
        }
}

p1 = Person.new("Lee", 22, "male", "123456");
p2 = Person.new("Rey", 20, "female", "654321");
p2.setAddress("267 N 4th Ring Middle Rd, Beijing");
p3 = Person.new("Olaf", 5, "male", "111111");
p4 = Person.new("Elsa", 24, "female", "222222");
p5 = Person.new("Anna", 23, "female", "333333");
people = {p1, p2, p3, p4, p5};
people[0].getUp();
people[0].haveBreakfast("egg and milk");
people[1].celebrateBirthday();
people[3].compareAge(people[4]);
```

运行结果均为先输入源代码，如果没有语法错误，将会输出

```
>>>>>>>>>>no error!<<<<<<<<<<<<<
```

我们设计了UI，展示效果为：



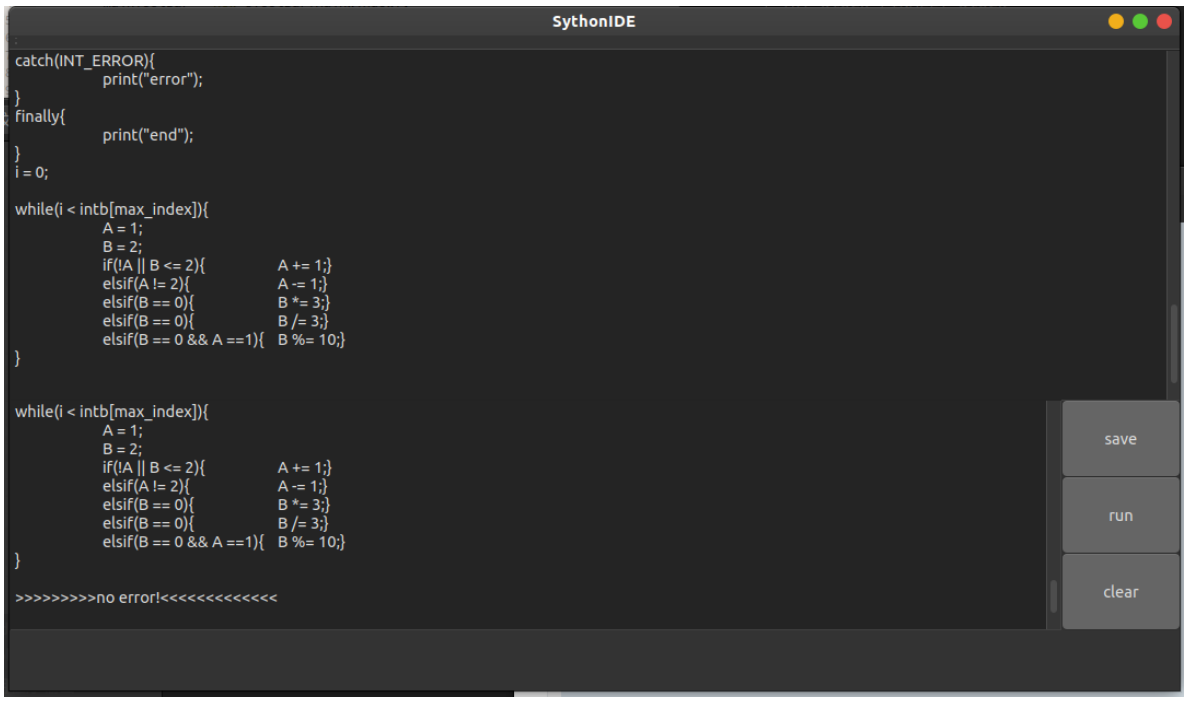
代码4

```
max_index = 9;
class a{
```

```
    attr_a;
    new(_attr_a){
        attr_a = _attr_a;
    }

    aprint(){
        print("aaaa");
    }
}

class b extend a{
    attr_b,attr_2;
    new(_attr_b, _attr_2){
        attr_b = _attr_b;
    }

    bprint(){
        print("bbbb");
    }
}

class tEst_I{
    de1f;
    new(_de1f){
        de1f = _de1f;
    }
}

class TeSt_INde extend tEst_I{
    attr_a,attr_2;
    new(_attr_a, _attr_2){
        attr_a = _attr_b;
    }
    a(){
        print("aaaa");
    }
}
```

```
print("hhh");
intb = {0,1,2,3,4,5,6,7,8,9};
try{
    for( i = 0; i < intb[max_index]; i++){
        if(i %2 == 0){
            print(intb[i]);
        }
        else{
            print("mod!=0");}
    }
}
catch(INT_ERROR){
    print("error");
}
finally{
    print("end");
}
i = 0;

while(i < intb[max_index]){
    A = 1;
    B = 2;
    if(!A || B <= 2){   A += 1;}
    elsif(A != 2){  A -= 1;}
    elsif(B == 0){  B *= 3;}
    elsif(B == 0){  B /= 3;}
    elsif(B == 0 && A ==1){ B %= 10;}
}
```

同上，语法正确



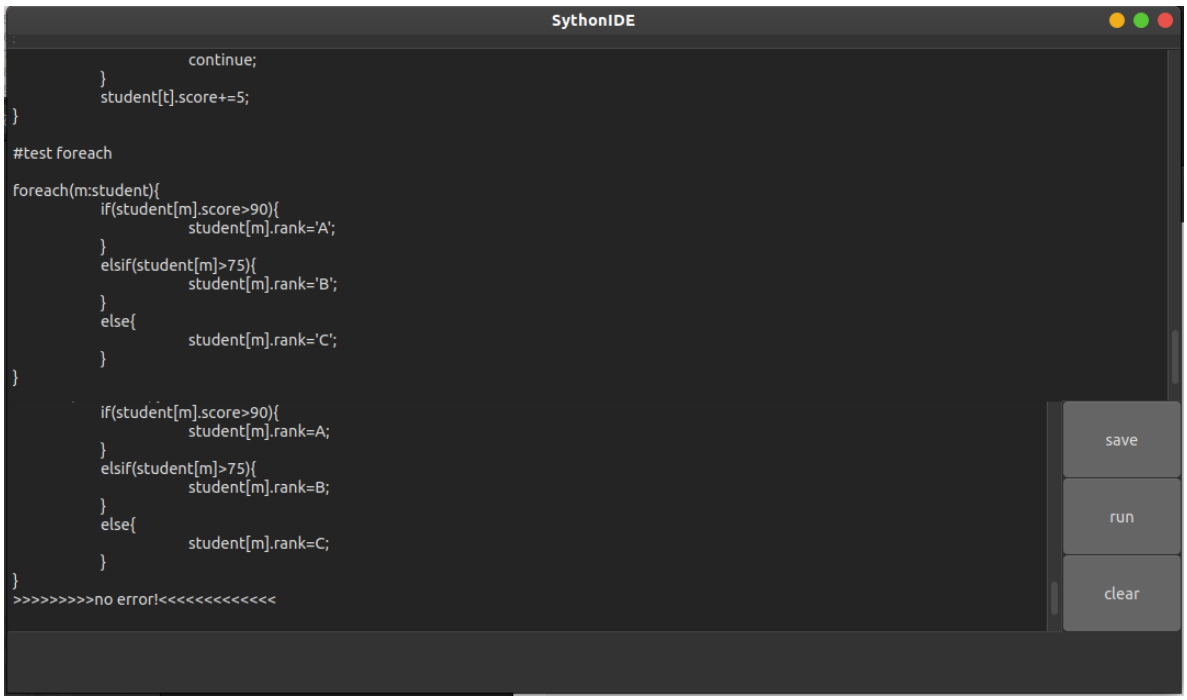
代码5

```
#test function defination
function isPrime(num){
    tmp=num-1;
    for(i=2;i<tmp;i=i+1){
        if(num%i==0){
            print("no");
            return false;
        }
    }
    print("yes");
    return true;
}

#test function call
for(j=1;j<100;j=j+1){
    if(isPrime(j)){
        print(j+"is Prime");
    }
}

#test class def

class Person{
```

```
    name,age,address;
}
class Student extend Person{
    sid,
    classroom = "101",
    score,
    rank;

    new(name1,sid1,score1){
        name=name1;
        sid=sid1;
        score=score1;
    }


    show_score(){
        print(name+"'s score is"+score);
    }


}
#test return
function max(num1,num2){
    if(num1>num2){
        return num1;
    }
    else{
        return num2;
    }

}

#test class call

stu1=Student.new("1","Kiki",99);
stu2=Student.new("2","Chuchu",74);
stu3=Student.new("3","Lily",93);
stu4=Student.new("4","Dode",66);
```

```
student[5]={stu1,stu2,stu3,stu4};

maxScore=stu1.score;
topId=1;

#test for
for(k=1;k<4;k++){
    if(student[k]>max){
        maxScore=student[k];
        topId=k;
    }
}
print(topId+"get the max score:"+maxScore);

#test while
t=0;
while(t<4){
    if(student[t].score>=98){
        continue;
    }
    student[t].score+=5;
}

#test foreach

foreach(m:student){
    if(student[m].score>90){
        student[m].rank='A';
    }
    elsif(student[m]>75){
        student[m].rank='B';
    }
    else{
        student[m].rank='C';
    }
}
```

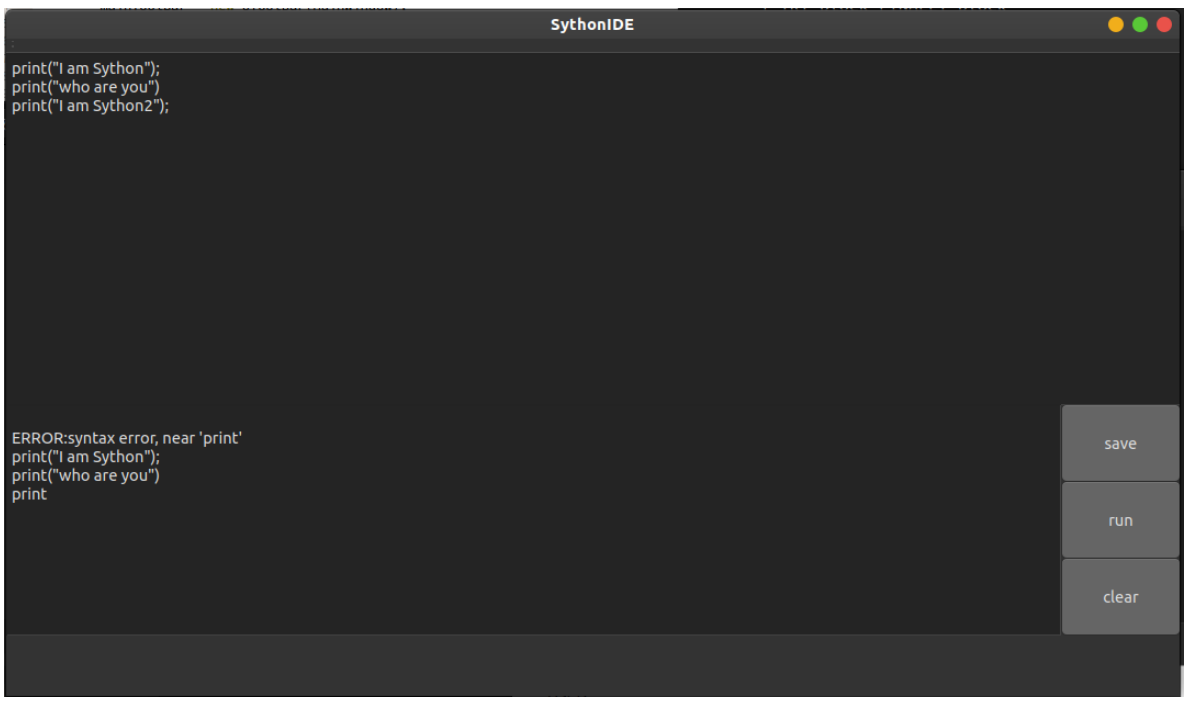
运行效果如图



代码6

```
print("I am Sython");
print("who are you")
print("I am Sython2");
```

运行结果如下，缺少一个";"，将会提示语法错误

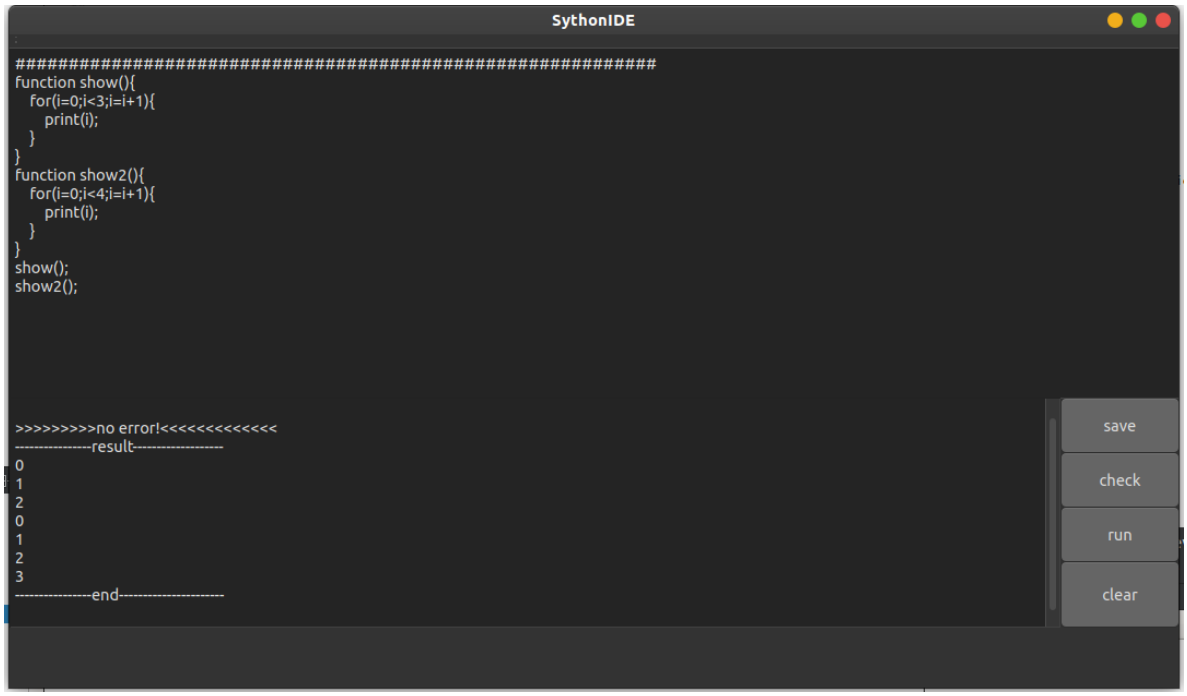


以下部分的代码可在最终完成的解释器上运行

代码7

```
#############################################################
#####
function show(){
    for(i=0;i<3;i=i+1){
        print(i);
    }
}
function show2(){
    for(i=0;i<4;i=i+1){
        print(i);
    }
}
show();
show2();
```

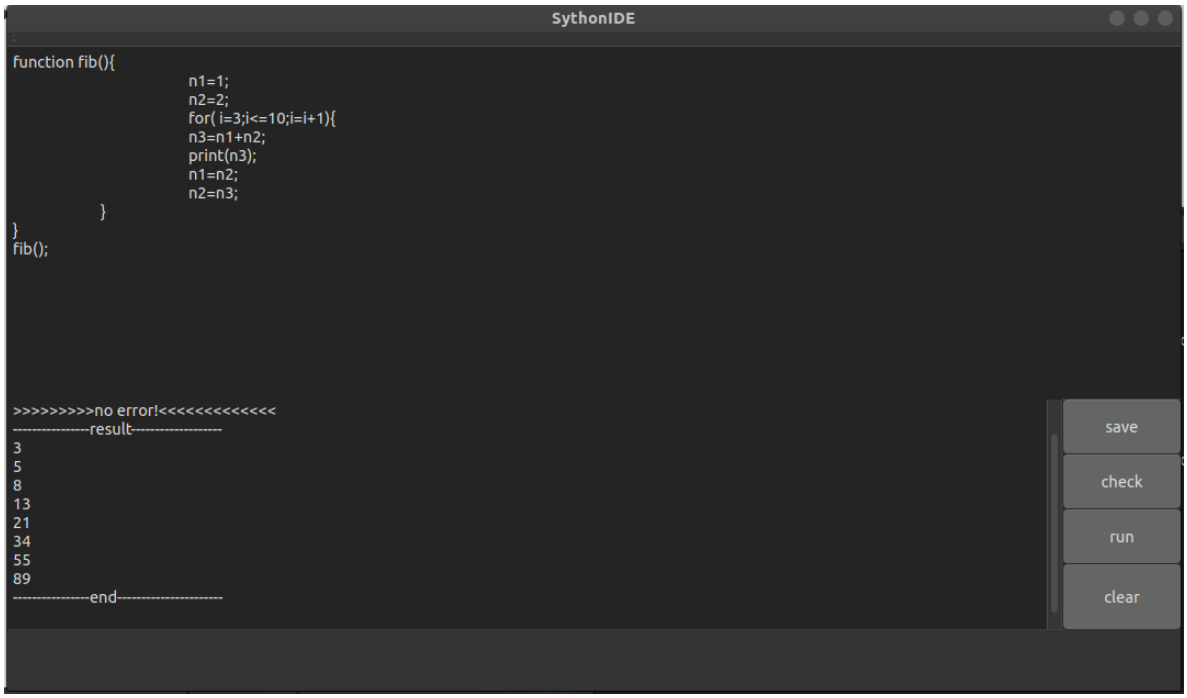
运行结果为，



代码8

```
function fib(){
        n1=1;
        n2=2;
        for( i=3;i<=10;i=i+1){
            n3=n1+n2;
            print(n3);
            n1=n2;
            n2=n3;
    }
}
fib();
```

运行结果：



解释器运行范围说明

语法分析器

语法分析器可以对文法规定范围内的文法进行语法检查，具体执行文件在/interpreter/可执行文件夹内，文档内readme指明了各个执行文件的运行方式。

解释器

目前解释器完成了部分执行功能，支持的操作包括：

- 四则运算（不包括++、--等自增运算）
- for循环
- print常量以及变量
- 变量赋值
- if语句
- 函数调用（不支持函数传参）

文档说明

目录结构

在interpreter文件夹下有两个文件夹

--interpreter

----UI 包含Ubuntu环境下图形化界面文件以及源码

----可执行文件 将四个可执行文件拿出来了，Ubuntu、Windows下运行对应程序即可，运行方式见下

运行方式

Ubuntu

进入终端，cd到`Sython`文件所在目录，运行。`.\Sython code.txt`

Windows

打开可执行文件`SythonWin.exe`，程序提示输入代码路径，输入路径即可

IDE运行方式（仅限Ubuntu）

进入UI目录，输入`.\SythonIDE`,即可打开界面

在输入框输入代码，点击`save`,提示保存成功，然后点击`check`可以检查语法，点击`run`可以运行，点击`clear`可以清除输入和输出内容。

其他

Sython全部源代码已经上传至github。

参考资料说明

Sython基于C语言文法的一个子集进行扩充，本小组自行构建完成了词法分析和语法分析，并建立了语法树，后使用开源的语法树执行平台接口进行了完成了部分完成了解释器，在此过程中，对原后端程序进行了数据结构上的调整和接口的重新封装。

[1] C语言基本文法

[2] https://github.com/yangtau/hedgehog