# Jython

## سند زبان جایتون

نویسنده : نجمه حبیبی

ایمیل : nh13nh76@yahoo.com

## مقدمه

جایتون یک زبان برنامه نویسی شی گراست که از تلفیق جاوا و پایتون ساخته شده است. البته جایتون تمام قابلیت های این دو زبان را ندارد. هر برنامه جایتون از تعدادی فایل با پسوند **.jy** تشکیل می شود. در ادامه به بررسی قابلیت های زبان جایتون می پردازیم:

## تعریف کلاس

در هر فایل امکان تعریف حداکثر یک کلاس وجود دارد. نام کلاس باید با حروف بزرگ شروع شود . هر کلاس شامل خصیصه ، متد و سازنده کلاس است . نحوه تعریف کلاس ها به شکل زیر است :

```
class class_name {
    #class_body
}
```

## اضافه کردن کلاس های از پیش تعریف شده

در صورت استفاده از سایر کلاس ها باید آنها را قبل از شروع کلاس تعریف (import) کنیم . برای مثال :

```
import Bread
import Tomato
class Sandwitch{
    Bread bread
    Tomato tomato
    int price
}
```

## وراثت

همانطور که قبلا اشاره کردیم زبان جایتون زبانی شی گراست. ارث بری در این زبان شباهت زیادی با زبان جاوا دارد.

وراثت به شکل زیر تعریف می شود :

```
class child_name (parent_name){
    #class_body
}
```

متد های کلاس پدر قابلیت override شدن در کلاس فرزند را دارند ، اما به طوری کلی الزامی برای override کردن وجود ندارد و میتوان از متد های پدر برای فرزند نیز استفاده کرد . همچنین تمامی خصیصه های موجود در کلاس پدر ، خصیصه کلاس فرزند نیز خواهند بود.

در زبان های برنامه نویسی شی گرا، قابلیت override متدها قابلیتی است که در آن یک کلاس فرزند میتواند پیاده سازی خاص خود را از متدهای کلاس پدر داشته باشد و در زمان اجرا آن پیاده سازی به جای پیاده سازی پدر اجرا خواهد شد.

وراثت چندگانه در این زبان تعریف نشده است ، همچنین وراثت نمی تواند به گونه ای باشد که دور به وجود بیاورد.

همه کلاس ها در صورتی که از یک کلاس ارث نبرد به صورت پیش فرض از کلاس 'Object' ارث می برند.

## سازنده کلاس

سازنده متدی است هم نام با کلاس که یک شی از کلاس بازمی گرداند . هر کلاس می تواند سازنده های متفاوتی داشته باشد.

```
class Sandwich{
   def Sandwich(HotDog hotdog, Bread bread , Tomato tomato){
        #Constructor body
   }
}
```

برای ساخت شی از یک کلاس سازنده آن را صدا می زنیم . نحوه ساختن شیء :

```
class Sandwich{
  def void test(){
   Bread bread = Bread()  #bread is an inatance of Bread class
  }
}
```

## خصیصه

هر کلاس دارای تعدادی خصیصه است . در زمان تعریف خصیصه حتما نوع آن باید تعیین شود . خصیصه ها فقط داخل متد ها مقدار می گیرند. خصیصه ها داخل کلاس تعریف می شوند :

```
class Human{
    Nose nose
    Hand[2] hand #this is an array
    Leg[2]  leg  #this is an array
}
```

## متغییر

متغییر ها مشابه خصیصه ها تعریف می شوند با این تفاوت که متغییر ها داخل متد ها قابل تعریف هستند.

```
   def void my_method(){
        int counter = 0
        float i
   }
```

## انواع داده

در زبان جایتون انواع داده های زیر موجود هستند.

| int |
| --- |
| float |
| bool |
| string |
| class |

علاوه بر انواع گفته شده متغییر ها و خصیصه ها می توانند از نوع کلاس نیز باشند .

در این زبان یک نوع آرایه نیز تعریف شده است . این آرایه یک بعدی است و میتواند از هر گونه ای باشد. مثالی از تعریف ارایه:

```
int[2] array = int()[2]
Food[10] foods = Food()[10]
```

## متد

هر کلاس دارای تعدادی متد میتواند باشد. نوع مقداری که یک متد بازمی گرداند و همچنین نوع ورودی های آن باید حتما مشخص باشد. در صورتی که متد خروجی نداشته باشد از کلمه 'void' به عنوان نوع خروجی استفاده می کنیم. پارامتر ها با استفاده از ' , ' از یکدیگر جدا می شوند . نحوه تعریف متد در زبان jython به شکل زیر است :

```
def return_type method_name (<parameters>){
 #method_body
 return return_vlaue
}
```

عمل return در هر قسمت از متد امکان دارد انجام شود و هیچ محدودیتی برای تعداد تعریف return وجود ندارد .

در زبان جایتون کد های نوشته شده بعد از return اجرا نمی شوند .

در صورتی که متد خروجی نداشته باشد نیاز به نوشتن 'return' نمی باشد. مثال:

```
class Human{
    def Voice speak(){
        #this method returns something
        return voice
    }
    def void eat(Food food){
        #this method dosen't return something
    }
}
```

## متد اصلی

هر برنامه ای باید شامل فقط یک متد اصلی باشد . متد اصلی در هر کلاسی می تواند تعریف شود .

متد اصلی مانند متدی است با نام 'main' که مقداری باز نمی گرداند. همچنین متد اصلی ورودی ندارد.

```
def void main (){
   #code
}
```

## ساختار تکرار

این زبان شامل دو مدل ساختار تکرار  است :

### ۱. ساختار while

```
while(<conditions>){
    #loop body
}
```

### ۲. ساختار for

این ساختارخود نیز انواع مختلفی دارد . به مثال های زیر توجه کنید :

```
for item in array{
    print (item)
}
```

```
for i in range(6){  #output 0 1 2 3 4 5
    print (i)
}
```

```
for i in range(1,5){  # output: 1 2 3 4
    print (i)
}
```

```
for i in range(1,10,2){  # output: 1 3 5 7 9
    print (i)
}
```

## ساختار تصمیم گیری

در زبان جایتون تنها ساختار تصمیم گیری ساختار شرطی است که با if شروع می شود و میتواند چند elif داشته باشد و در نهایت با else به پایان می رسد. همچنین ساختار if میتواند بدون elif و else مورد استفاده قرار گیرد.

```
if(<conditions>){
    #code
}elif(<conditions>){
    #code
}else{
    #code
}
```

## کامنت

در زبان جایتون دو نوع کامنت وجود دارد :

۱. کامنت تک خطی : تمامی حروف بعد از # تا انتهای خط جزو کامنت تک خطی می باشند .
۲. کامنت چندخطی : تمامی حروف بعد از #* تا زمانی که *# دیده شود جز کامنت چند خطی می باشند.

به قطعه کد زیر توجه کنید :

```
class Test{
    def void comments(){
        #single-line comment
        print("")
        #*
             multi-line comment
             multi-line comment
        *#
    }
}
```

دقت کنید که در زبان جایتون کامنت ساختار یافته یا تودرتو وجود ندارد .

## کلیدواژه self

واژه self به کلاسی که در آن قرار داریم اشاره می کند که به وسیله آن می توان به خصیصه یا متد کلاس دسترسی پیدا کرد.

```
def void human (){
   self.eat(food);
}
```

## متد print

متد print به صورت ضمنی تعریف شده است و می تواند مقادیرbool , string , int را در کنسول چاپ کند.

برای مثال:

```
print(10)                # output: 10
print("hello")           # output: hello
print(human.name)        # output: Ali
print(human.getcount()) # output: 15
print(foods[1])          # output: sandwich
print(true)              # output: true
print(say_hi())          # output: hi
```

در صورتی که داخل متد print یک متد صدا زده شود مقدار خروجی متد چاپ می شود .

## Scope ها

در زبان جایتون scope با کاراکتر '{' شروع می شود و با '}' پایان می یابد.

کلاس ها ، متد ها ، ساختار های تکرار و ساختار های تصمیم گیری هرکدام scope هستند.

متغییر ها ، خصیصه ها و متد ها فقط در داخل یک scope قابل دسترسی هستند. بنابراین اگر در scope فرزند متغییری تعریف شده باشد در scope پدر قابل دسترس نخواهد بود ، اما اگر در scope پدر متغییری تعریف شده باشد در scope فرزند قابل دسترس است.

به مثال زیر توجه کنید :

```
class Human{
    bool isHungry
    def void eat(Food food){
        newFood = food   #ERROR: newFood is not defined
        while(self.isHungry){
            Food newFood = Food()
            eat(newfood)
            self.isHungry = self.checkIsHungry();
        }
    }
    def bool checkIsHungry(){
        return true;
    }
}
```

## کلاس های تعریف شده توسط خود زبان جایتون

زبان جایتون مانند سایرزبان ها دارای کلاس هایی است که در کتابخانه خود زبان موجود هستند.

زبان جایتون شامل کلاس های Object, String, Math, Random است که هر یک را جداگانه تعریف می کنیم:

| Class name | methods | Definition |
|---|---|---|
| Object | | |
| String | concat(string s1,string s2) | concatenates s2 to the end of s1 and return new string. |
| | length() | returns the length of this string. |
| | substring(int beginIndex, int endIndex) | returns a new string that is a substring of this string. |
| Math | max(int n1,int n2) | returns the largest of two values. |
| | min(int n1,int n2) | returns the smallest of two values. |
| Random | get_int() | returns a random number. |
| | get_int(int n1 ,int n2) | returns a random number between n1 and n2. |

توجه کنید که کلاس String با نوع داده string متفاوت است .

## کلمات کلیدی زبان جایتون

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **while** | **if** | **elif** | **else** | **for** | **and** | **or** | **class** |
| **def** | **return** | **import** | **none** | **true** | **false** | **void** | **print** |
| **range** | **float** | **int** | **bool** | **string** | | | |

## عملگرها

درصورتی که A=10 و B=20 عملگر های زبان جایتون را به صورت زیر تعریف می کنیم:

### عملگر های ریاضی :

| Operator | Description | Example |
|---|---|---|
| + (Addition) | Adds values on either side of the operator. | A + B will give 30 |
| - (Subtraction) | Subtracts right-hand operand from left-hand operand. | A - B will give -10 |
| * (Multiplication) | Multiplies values on either side of the operator. | A * B will give 200 |
| / (Division) | Divides left-hand operand by right-hand operand. | B / A will give 2 |
| % (Modulus) | Divides left-hand operand by right-hand operand and returns remainder. | B % A will give 0 |

### عملگر های انتساب :

| Operator | Description | Example |
|---|---|---|
| = | Simple assignment operator. Assigns values from right side operands to left side operand. | C = A + B will assign value of A + B into C |
| += | Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand. | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand. | C -= A is equivalent to C = C – A |
| *= | Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand. | C *= A is equivalent to C = C * A |
| /= | Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand. | C /= A is equivalent to C = C / A |

عملگر های مقایسه ای :

| Operator | Description | Example |
|---|---|---|
| == (equal to) | Checks if the values of two operands are equal or not, if yes then condition becomes true. | (A == B) is not true. |
| != (not equal to) | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | (A != B) is true. |
| > (greater than) | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (A > B) is not true. |
| < (less than) | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (A < B) is true. |
| >= (greater than or equal to) | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true. |
| <= (less than or equal to) | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (A <= B) is true. |

**اولویت عملگر ها :**

| اولویت | عملگر ها | دسته | شرکت پذیری |
|---|---|---|---|
| ۱ | **( )** | **پرانتز** | **چپ** |
| ۲ | **[ ]**<br>**.** | **دسترسی به عناصر آرایه ، دسترسی به خصیصه ، فراخوانی متد یک شی** | **چپ** |
| ۳ | **/ * %** | **ضرب ، تقسیم و باقی مانده** | **چپ** |
| ۴ | **- +** | **جمع و تفریق** | **چپ** |
| ۵ | **< ، > ، =< ، =>** | **رابطه ای** | **چپ** |
| ۶ | **== ، =!** | **مقایسه ی تساوی** | **چپ** |
| ۷ | **And or** | **عطف و فصل منطقی** | **چپ** |
| ۸ | **= ,+=,-=,*=,/=** | **تخصیص** | **راست به چپ** |
| ۹ | **,** | **ورودی متد ها** | **چپ به راست** |

**گرامر**

```
program : importclass* (classDef)?   ;

importclass : 'import' CLASSNAME ;

classDef :  'class' CLASSNAME ('(' CLASSNAME ')')? '{' class_body* '}';

class_body :  varDec    | methodDec  | constructor    | arrayDec  ;

varDec :   TYPE  ID  ;

arrayDec : TYPE '['INTEGER']' ID  ;

methodDec : 'def'  (TYPE|'void')   ID  '(' parameter* ')''{' ( statement)* '}';

constructor : 'def'  TYPE '(' parameter* ')''{' ( statement)* '}' ;

parameter : varDec (',' varDec)* ;

statement : varDec  | assignment   | print_statment  | method_call  | return_statment
        |if_statment    | while_statment   | if_else_statment   | for_statment;

return_statment : 'return' exp ;

condition_list : condition (('or'|'and') condition)* ;

condition : BOOL  | prefixexp    | (exp) relational_operators (exp);

if_statment : 'if' '(' condition_list ')' '{' statement* '}';

while_statment : 'while' '(' condition_list ')' '{' statement* '}' ;

if_else_statment :'if' '(' condition_list ')' '{' statement* '}' ('elif' '(' condition_list ')' '{' statement*
'}')* 'else' '{' statement* '}'  ;

print_statment : 'print' '(' (prefixexp | TYPE args | INTEGER |STRING | BOOL) ')' ;

for_statment : 'for' ID 'in' ID '{' statement* '}'
                | 'for' ID 'in' 'range''('INTEGER (',' INTEGER)? (',' INTEGER)? ')' '{' statement* '}' ;

method_call : ID '.'args  ;

assignment : prefixexp assignment_operators exp
            | varDec assignment_operators exp
            | arrayDec '='  TYPE args ('['INTEGER']') ;

exp :'none'  | BOOL  | INTEGER  | STRING  | FLOAT  | prefixexp  | exp arithmetic_operator exp
      | TYPE args  | '('exp')'   | ID args ;
```

```
prefixexp : ID  | prefixexp '[' INTEGER ']'  | prefixexp '.' ID  | prefixexp '.' ID args  ;

args :  '(' (explist)? ')' ;

explist : exp (',' exp)*;

arithmetic_operator:  '+'  |  '-'  | '*'  |  '/'  | '%'  ;

relational_operators :  '<' |  '>'  | '<='  | '>='  | '=='  | '!=' ;

assignment_operators :  '='  | '+='  | '-='  | '*=' | '/=' ;
```

**توضیحات گرامر :**

*A به معنای صفر یا تعداد بیشتر A می باشد . +A به معنای یک یا تعداد بیشتر A می باشد .

? به معنای اختیاری بودن است.

CLASSNAME : رشته هایی شامل نام کلاس که حتما باید با حروف بزرگ  شروع شوند.

TYPE: رشته های شامل حروف و ارقام که حتما با حروف بزرگ آغاز شوند. نام کلاس ها از نوع TYPE هستند. علاوه بر این انواع داده های زبان (int,float,string,…) نیز TYPE هستند.

ID : رشته هایی شامل ’_‘ و حروف و ارقام که با حروف کوچک آغاز شوند . نام متد ها ، متغییر ها و خصیصه ها از نوع ID است.