A picture containing text

Description automatically generated

**دانشکده مهندسی کامپیوتر**

گزارش فاز دوم پروژه کامپایلر - گروه ۶

**اعضای گروه**

باوان دیوانی آذر - بیان دیوانی آذر - پارمیدا مجمع صنایع - زهرا مومنی نژاد

**تیر ۱۴۰۱**

فهرست مطالب:

[تسک‌ها: 3](#_Toc107698626)

[تقسیم‌بندی: 3](#_Toc107698627)

[روش انجام: 3](#_Toc107698628)

[تسک اول – CountDeclMethodAll 3](#_Toc107698629)

[تسک دوم – CountDeclMethodDefault 9](#_Toc107698630)

[تسک سوم – CountDeclMethodPrivate 10](#_Toc107698631)

[تسک چهارم – CountDeclMethodProtected 12](#_Toc107698632)

[\* خروجی گرفتن با استفاده از تابع‌های خودمان 13](#_Toc107698633)

[\* خروجی گرفتن از understand 14](#_Toc107698634)

[مقایسه خروجی‌ها 15](#_Toc107698635)

# **تسک‌ها:**

تسک‌های گروه ما به شرح زیر بود:

* 1. CountDeclMethodAll
  2. CountDeclMethodDefault
  3. CountDeclMethodPrivate
  4. countDeclMethodProtected

# **تقسیم‌بندی:**

نحوه انجام کار به این صورت بود که ۲ تسک اول را خانم باوان دیوانی آذر و بیان دیوانی اذر انجام دادند و ۲ تسک بعدی را خانم پارمیدا مجمع صنایع و زهرا مومنی نژاد انجام دادند. البته تمرکز بیشتر هر نفر به این صورت بوده و همگی در انجام همه‌ی موارد نقش داشتند.

# **روش انجام:**

در فاز ۲ ما باید خروجی موارد بالا طبق دیتابیس understand و api های آن را با خروجی موارد بالا طبق دیتابیس خودمان و توابعی که برای کوئری زدن می‌زنیم مقایسه کنیم. همان طور که می‌دانیم دیتابیس ذکر شده در فایل main ایجاد شده و طبق فایل‌های پیاده‌سازی شده در فاز ۱ که در پوشه‌ی analysis passes هستند، پر می‌شود.

حال مفهوم تسک‌ها و نحوه پیاده‌سازی آن‌ها را بررسی می‌کنیم.

# **تسک اول – CountDeclMethodAll**

**ابتدا مفهوم این بخش را شرح می‌دهیم.**

منظور از این قسمت تعداد همه‌ی توابع یک فایل می‌باشد. که شامل تمام توابع خود فایل و تمام توابع فایل‌هایی که فایل ما از آن ارث‌بری کرده است می‌باشد. برای فهم بهتر مثال زیر را بررسی می‌کنیم:

یک کلاس employee داریم و یک کلاس manager که از آن ارث‌بری می‌کند. حال می‌خواهیم تعداد توابع کل manager را محاسبه کنیم. برای اینکار باید هم توابع خود فایل manager را درنظر بگیریم، و هم توابع پدر آن را. یعنی کلاس employee.

فایل employee که یک کلاس دارد به شرح زیر است:

public class employee {  
   
 static long employeeId;  
 static String employeeName;  
 static String employeeAddress;  
 static long employeePhone;  
 static double basicSalary;  
 static double specialAllowance = *250.80*;  
 static double hra = *1000.50*;  
   
 default static long getEmployeeId() {  
 return employeeId;  
 }  
 protected static void setEmployeeId(long Id) {  
 employee.employeeId = Id;  
 }

protected static String getEmployeeName() {  
 return employeeName;  
 }  
 default static void setEmployeeName(String Name) {  
 employee.employeeName = Name;  
 }  
 protected static String getEmployeeAddress() {  
 return employeeAddress;  
 }  
 protected static void setEmployeeAddress(String address) {  
 employee.employeeAddress = address;  
 }  
 default static long getEmployeePhone() {  
 return employeePhone;  
 }  
 protected static void setEmployeePhone(long phone) {  
 employee.employeePhone = phone;  
 }  
  
 private static void calculateSalary () {   
 double salary = basicSalary + (basicSalary \* specialAllowance/100) + basicSalary \* hra/100;  
 System.out.println(salary);  
 }  
   
 private double calculateTransportAllowance () {  
 double transportAllowance = 10/100\*basicSalary;   
 return transportAllowance;  
 }  
   
}

تعداد کل تابع‌های این فایل ۱۰ تاست. حال فایل manager را بررسی کنیم:

public class Manager extends employee {  
 protected static double getBasicSalary() {  
 return basicSalary;  
 }  
 protected static void setBasicSalary(double basicSalary) {  
 Manager.basicSalary = basicSalary;  
 }  
 @Override  
 private double calculateTransportAllowance () {  
 double transportAllowance = *15* \* basicSalary/100;   
 return transportAllowance;  
 }

تعداد کل تابع‌های این فایل هم ۳ تاست. اما چون این کلاس از کلاس employee ارث‌بری کرده است، باید تعداد توابع کلاس پدر را هم در تعداد توابع کل manager حساب کنیم. پس CountDeclMethodAll برابر با **۱۳** تا خواهد بود.

**حال به نحوه پیاده‌سازی این بخش می‌پردازیم:**

ابتدا برای اینکار یک فایل پایتون متناسب با اسم تسک در پوشه metrics ایجاد کردیم.

در ابتدا فایل‌های مورد نیاز و توابع مورد نیاز را import کردیم و بعد هم تابع walk را برای پیمایش روی درخت تعریف کردیم.

from oudb.api import open  
from oudb.models import EntityModel, KindModel  
from antlr4 import \*  
from gen.javaLabeled.JavaParserLabeled import JavaParserLabeled  
from gen.javaLabeled.JavaLexer import JavaLexer  
from analysis\_passes.extend\_listener\_g6 import ExtendListener  
  
  
def Walk(reference\_listener, parse\_tree):  
 walker = ParseTreeWalker()  
 walker.walk(listener=reference\_listener, t=parse\_tree)

سپس یک تابع به نام count\_decl\_method\_all ایجاد کردیم.

در آن دو دیکشنری به نام‌های class\_methods و extend\_class\_names تعریف کردیم. که به ترتیب اولی نام فایل و تعداد توابع کل و دومی نام کلاس و نام کلاس‌هایی که کلاس ما از آن‌ها ارث‌بری می‌کند را نگه می‌دارد.

یک لیست به نام files هم داریم که فایل‌ها را نگه می‌دارد.

سپس یک حلقه for زدیم که چک کنیم در بین entity model ها هر کدام که فایل بود را به لیست خود اضافه می‌کنیم. از این طریق به فایل‌های پروژه دسترسی پیدا می‌کنیم. و اگر class بود دیکشنری class\_methods را با key با ارزش اسم entity و value با ارزش ۰ اپدیت می‌کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

def count\_decl\_method\_all(dbname):  
 open(dbname)  
 class\_methods = {}  
 files = []  
 extends\_class\_names = {}  
 *# get files names* for ent\_model in EntityModel.select():  
 if ent\_model.\_kind\_id == 1:  
 files.append(ent\_model.\_longname)  
 if "Class" in ent\_model.\_kind.\_name:  
 class\_methods[ent\_model.\_name]=0

در مرحله‌ی بعدی، یک حلقه‌ی for دیگر برای کال کردن listener زده شده با عنوان extendListener استفاده کردیم. که این listener را در فایل extend\_listener\_g6 تعریف کردیم و اکنون باهم عملکرد آن را بررسی کرده و به ادامه این بخش بازمی‌گردیم.

در این فایل ابتدا دو فایل parser و listener مورد نیاز خود را import کردیم و یک کلاس جدید برای listener خود تعریف کردیم. که این کلاس ۲ تا property دارد. یک class\_name که اسم کلاس است و یک دیکشنری به نام refers که روابط و parent ها و ارث‌بری‌ها را نگه می‌دارد. همچنین دو تابع getter برای دریافت مقادیر این دو متغیر تعریف کردیم.

سپس قسمت اصلی اینجاست که ما تابع enterClassDeclaration را override کردیم. برای چه؟ برای اینکه پیدا کنیم هر کلاس چه کلاس‌هایی را extend کرده است و آن‌ها را در دیکشنری خود ذخیره کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

from gen.javaLabeled.JavaParserLabeledListener import JavaParserLabeledListener  
from gen.javaLabeled.JavaParserLabeled import JavaParserLabeled  
  
class ExtendListener(JavaParserLabeledListener):  
  
 def \_\_init\_\_(self):  
 self.class\_name = None  
 self.refers = {}  
  
  
 def get\_refers(self):  
 return self.refers

def get\_class\_name(self):  
 return self.class\_name  
def enterClassDeclaration(self, ctx:JavaParserLabeled.ClassDeclarationContext):  
 self.class\_name = ctx.IDENTIFIER().getText()  
 if ctx.getChild(2).getText() == "extends":  
 childs=ctx.getChild(3).getChildren()  
 for c in childs:  
 if not self.refers.\_\_contains\_\_(self.class\_name):  
 self.refers[self.class\_name] = []  
 self.refers[self.class\_name].append(c.getText())

توضیح و عملکرد extendListener را با هم دیدیم. به ادامه‌ی توضیح کلاس count\_decl\_method\_all برمی‌گردیم. همانطور که بیان شد، از یک حلقه‌ی for استفاده می‌کنیم تا روی تمامی فایل‌های پروژه این extendListener را صدا بزنیم و بعد روی درخت پیمایش می‌کنیم تا تابع override شده انجام شود و دیکشنری refers در extendListener پر شود. بعد آن را get می‌کنیم و در دیکشنری extend\_class\_names ذخیره می‌کنیم. تا اینجا لیست ارث‌بری‌ها را بدست آوردیم.

کد این قسمت توضیح داده شده به صورت زیر است:

*# get parent names*for file\_address in files:  
 try:  
 file\_stream = FileStream(file\_address, encoding='utf8')  
 lexer = JavaLexer(file\_stream)  
 tokens = CommonTokenStream(lexer)  
 parser = JavaParserLabeled(tokens)  
 parse\_tree = parser.compilationUnit();  
 except Exception as e:  
 print("An Error occurred in file:" + file\_address + "\n" + str(e))  
 continue  
 try:  
 listener = ExtendListener()  
 Walk(listener,parse\_tree)  
 extends\_class\_names.update(listener.get\_refers())  
 except Exception as e:  
 print("An Error occurred for reference implement in file:" + file\_address + "\n" + str(e))

حال می‌خواهیم توابع را پیدا کنیم. برای همین از یک حلقه for‌ استفاده می‌کنیم. و روی entity model ها پیمایش می‌کنیم و درصورتی که kind name آن برابر Method باشد، دیکشنری class\_method خود را اپدیت می‌کنیم. به این صورت که key اسم parent باشد و value هم یک عدد است که تعداد توابع را نشان می‌دهد. و اگر initial شده باشد هر بار count آن درصورت یافتن Method جدید یکی زیاد می‌شود.

کد این قسمت توضیح داده شده به صورت زیر است:

*# get class methods number*for ent\_model in EntityModel.select():  
  
 if "Method" in ent\_model.\_kind.\_name:  
 exists = class\_methods.get(ent\_model.\_parent.\_name, -1)  
 if exists == -1:  
 class\_methods[ent\_model.\_parent.\_name] = 1  
 else:  
 class\_methods[ent\_model.\_parent.\_name] += 1

در آخر هم از یک حلقه for استفاده می‌کنیم تا روی extend\_class\_names پیمایش کنیم و یک لیست visited و یک stack تعریف می‌کنیم. برای اینکه بیایم چک کنیم که کلاسی که هر کلاس از آن ارث بری کرده است آیا خودش هم از کلاسی ارث‌بری کرده است. و بعد آن را به class\_methods اضافه می‌کنیم. و در آخر تابع آن را return می‌کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

for cm in class\_methods:  
 stack=[]  
 visited=[]  
 temp = cm  
 while extends\_class\_names.\_\_contains\_\_(temp):  
 for t in extends\_class\_names[temp]:  
 if not visited.\_\_contains\_\_(t):  
 stack.append(t)  
 visited.append(t)  
 temp = stack.pop()  
 for v in visited:  
 class\_methods[cm] += class\_methods[v]  
  
  
return class\_methods

# **تسک دوم – CountDeclMethodDefault**

**ابتدا مفهوم این بخش را شرح می‌دهیم.**

منظور از این بخش تعداد تمام توابعی هستند که با کلیدواژه‌ی default تعریف شدند. مثالی از این نوع تابع بصورت زیر است:

default void show()  
{  
 System.out.println("Default Method Executed");  
}

**حال به نحوه پیاده‌سازی این بخش می‌پردازیم:**

ابتدا برای اینکار یک فایل پایتون متناسب با اسم تسک در پوشه metrics ایجاد کردیم.

سپس از api های oudb مورد open و از model ها مورد EntityModel را import‌ کردیم. و بعد یک تابع به اسم تسک ایجاد کردیم که به عنوان ورودی مسیر دیتابیس ما را می‌گیرد. و بعد با استفاده از open آن را می‌خواند. یک دیکشنری به اسم class\_methods نیز تعریف کردیم که اسم هر کلاس را به عنوان کلید و تعداد توابع default آن را به عنوان value دربردارد.

چون می‌خواهیم توابع یک کلاس را بررسی کنیم، ابتدا روی entity model ها با یک حلقه for‌ پیمایش می‌کنیم و اگر kind name برابر با class بود دیکشنری را با اسم entity model به عنوان key و 0 به عنوان value آن initial می‌کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

from oudb.api import open  
from oudb.models import EntityModel  
  
  
def count\_decl\_method\_default(db\_path):  
 open(db\_path)  
 class\_methods = {}  
  
 for ent\_model in EntityModel.select():  
 if "Class" in ent\_model.\_kind.\_name:  
 class\_methods[ent\_model.\_name]=0

سپس دوباره برای پیدا کردن توابع default از یک حلقه for و از یک شرط استفاده می‌کنیم. به این صورت که چک می‌کنیم درون kind name آن entity model هم واژه‌ی default باشد و هم واژه‌ی method. سپس اگر اسم parent آن entity model را درون دیکشنری خود داشتیم، تعداد توابع آن کلاس را اپدیت می‌کنیم و یکی زیاد می‌کنیم. و در آخر تابع هم این دیکشنری را return می‌کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

for ent\_model in EntityModel.select():  
 if "Default" in ent\_model.\_kind.\_name and "Method" in ent\_model.\_kind.\_name:  
 exists = class\_methods.get(ent\_model.\_parent.\_name, -1)  
 if exists == -1:  
 class\_methods[ent\_model.\_parent.\_name] = 1  
 else:  
 class\_methods[ent\_model.\_parent.\_name] += 1  
return class\_methods

# **تسک سوم – CountDeclMethodPrivate**

**ابتدا مفهوم این بخش را شرح می‌دهیم.**

منظور از این بخش تعداد تمام توابعی هستند که با کلیدواژه‌ی private تعریف شدند. مثالی از این نوع تابع بصورت زیر است:

private double calculateTransportAllowance () {  
 double transportAllowance = 15 \* basicSalary/100;   
 return transportAllowance;  
}

**حال به نحوه پیاده‌سازی این بخش می‌پردازیم:**

ابتدا برای اینکار یک فایل پایتون متناسب با اسم تسک در پوشه metrics ایجاد کردیم.

سپس از api های oudb مورد open و از model ها مورد EntityModel را import‌ کردیم. و بعد یک تابع به اسم تسک ایجاد کردیم که به عنوان ورودی مسیر دیتابیس ما را می‌گیرد. و بعد با استفاده از open آن را می‌خواند. یک دیکشنری به اسم class\_methods نیز تعریف کردیم که اسم هر کلاس را به عنوان کلید و تعداد توابع private آن را به عنوان value دربردارد.

چون می‌خواهیم توابع یک کلاس را بررسی کنیم، ابتدا روی entity model ها با یک حلقه for‌ پیمایش می‌کنیم و اگر kind name برابر با class بود دیکشنری را با اسم entity model به عنوان key و 0 به عنوان value آن initial می‌کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

from oudb.api import open  
from oudb.models import EntityModel  
  
def count\_decl\_method\_private(db\_path):  
 open(db\_path)  
 class\_methods = {}  
  
 for ent\_model in EntityModel.select():  
 if "Class" in ent\_model.\_kind.\_name:  
 class\_methods[ent\_model.\_name]=0

سپس دوباره برای پیدا کردن توابع private از یک حلقه for و از یک شرط استفاده می‌کنیم. به این صورت که چک می‌کنیم درون kind name آن entity model هم واژه‌ی private باشد و هم واژه‌ی method. سپس اگر اسم parent آن entity model را درون دیکشنری خود داشتیم، تعداد توابع آن کلاس را اپدیت می‌کنیم و یکی زیاد می‌کنیم. و در آخر تابع هم این دیکشنری را return می‌کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

for ent\_model in EntityModel.select():  
 if "Private" in ent\_model.\_kind.\_name and "Method" in ent\_model.\_kind.\_name:  
 exists = class\_methods.get(ent\_model.\_parent.\_name, -1)  
 if exists == -1:  
 class\_methods[ent\_model.\_parent.\_name] = 1  
 else:  
 class\_methods[ent\_model.\_parent.\_name] += 1  
return class\_methods

# **تسک چهارم – CountDeclMethodProtected**

**ابتدا مفهوم این بخش را شرح می‌دهیم.**

منظور از این بخش تعداد تمام توابعی هستند که با کلیدواژه‌ی protected تعریف شدند. مثالی از این نوع تابع بصورت زیر است:

protected static void setBasicSalary(double basicSalary) {  
 Manager.basicSalary = basicSalary;  
}

**حال به نحوه پیاده‌سازی این بخش می‌پردازیم:**

ابتدا برای اینکار یک فایل پایتون متناسب با اسم تسک در پوشه metrics ایجاد کردیم.

سپس از api های oudb مورد open و از model ها مورد EntityModel را import‌ کردیم. و بعد یک تابع به اسم تسک ایجاد کردیم که به عنوان ورودی مسیر دیتابیس ما را می‌گیرد. و بعد با استفاده از open آن را می‌خواند. یک دیکشنری به اسم class\_methods نیز تعریف کردیم که اسم هر کلاس را به عنوان کلید و تعداد توابع protected آن را به عنوان value دربردارد.

چون می‌خواهیم توابع یک کلاس را بررسی کنیم، ابتدا روی entity model ها با یک حلقه for‌ پیمایش می‌کنیم و اگر kind name برابر با class بود دیکشنری را با اسم entity model به عنوان key و 0 به عنوان value آن initial می‌کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

from oudb.api import open  
from oudb.models import EntityModel  
  
  
def count\_decl\_method\_protected(db\_path):  
 open(db\_path)  
 class\_methods = {}  
  
 for ent\_model in EntityModel.select():  
 if "Class" in ent\_model.\_kind.\_name:  
 class\_methods[ent\_model.\_name]=0

سپس دوباره برای پیدا کردن توابع protected از یک حلقه for و از یک شرط استفاده می‌کنیم. به این صورت که چک می‌کنیم درون kind name آن entity model هم واژه‌ی protected باشد و هم واژه‌ی method. سپس اگر اسم parent آن entity model را درون دیکشنری خود داشتیم، تعداد توابع آن کلاس را اپدیت می‌کنیم و یکی زیاد می‌کنیم. و در آخر تابع هم این دیکشنری را return می‌کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

for ent\_model in EntityModel.select():  
 if "Protected" in ent\_model.\_kind.\_name and "Method" in ent\_model.\_kind.\_name:  
 exists = class\_methods.get(ent\_model.\_parent.\_name, -1)  
 if exists == -1:  
 class\_methods[ent\_model.\_parent.\_name] = 1  
 else:  
 class\_methods[ent\_model.\_parent.\_name] += 1  
return class\_methods

# **\* خروجی گرفتن با استفاده از تابع‌های خودمان**

برای خروجی گرفتن از ۴ تابعی که زدیم، یک فایل تست به نام db\_test\_metrics\_g6 ایجاد کردیم، که در آن ۴ تابع زده شده در ۴ فایل توضیح داده شده را import کردیم. یک تابع کلی به نام test\_all\_metrics زدیم که مسیر یک دیتابیس را به عنوان ورودی می گیرد و ۴ تابع برای پیدا کردن تعداد توابع all و default و private و protected را روی این ورودی صدا می‌زند.

کد این تابع بصورت زیر است:

from count\_decl\_method\_all import count\_decl\_method\_all  
from count\_decl\_method\_default import count\_decl\_method\_default  
from count\_decl\_method\_private import count\_decl\_method\_private  
from count\_decl\_method\_protected import count\_decl\_method\_protected  
  
def test\_all\_metrics(db\_path):  
 print("Our Results \n")  
 print("All methods : ", count\_decl\_method\_all(db\_path))  
 print("Default methods : ", count\_decl\_method\_default(db\_path))  
 print("Private methods : ", count\_decl\_method\_private(db\_path))  
 print("Protected methods : ", count\_decl\_method\_protected(db\_path))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 test\_all\_metrics("..\\..\\benchmark2\_database.oudb")

تابع را روی دیتابیس ساخته شده به نام benchmark2\_database.oudb اجرا می‌کنیم و خروجی به صورت زیر است:

A picture containing text

Description automatically generated

# **\* خروجی گرفتن از understand**

برای خروجی گرفتن از ابزار understand یک فایل تست در پوشه‌ی oudb به نام tests.py ایجاد کردیم. ابتدا موارد لازم برای اجرای تست را import کردیم و صدا زدیم. یک تابع به نام test\_understand\_kinds تعریف کردیم و با استفاده از open توسط und.open دیتابیس موردنظر را باز کردیم. برای نگه‌داری خروجی ۴ نوع metric مان ۴ آرایه برای توابع all و default و private و protected ایجاد کردیم.

کد این قسمت بصورت زیر است:

import os  
os.add\_dll\_directory(r"C:\Program Files\Scitools\bin\pc-win64")  
from dotenv import load\_dotenv  
load\_dotenv()  
try:  
 import understand as und  
except ImportError:  
 import understand as und  
  
 print("Can not import understand")  
  
def test\_understand\_kinds():  
 db = und.open("..\\..\\src.und")  
 und\_all\_results = {}  
 und\_default\_results = {}  
 und\_private\_results = {}  
 und\_protected\_results = {}

سپس با استفاده از یک حلقه‌ی for روی entity هایی که class‌ هستند پیمایش کردیم و از api آماده understand به نام metric استفاده کردیم و تعداد هر metric را بدست آوردیم. درنهایت هر ۴ مورد را print می‌کنیم.

کد این قسمت بصورت زیر است:

print("Understand Results \n")  
for ent in db.ents('Java Class ~Unknown ~Unresolved'):  
 ent\_name = ent.name()  
 all\_methods = ent.metric(['CountDeclMethodAll']).get('CountDeclMethodAll', 0)  
und\_all\_results[ent\_name] = all\_methods  
  
default\_methods = ent.metric(['CountDeclMethodDefault']).get('CountDeclMethodDefault', 0)  
und\_default\_results[ent\_name] = default\_methods  
private\_methods = ent.metric(['CountDeclMethodPrivate']).get('CountDeclMethodPrivate', 0)  
und\_private\_results[ent\_name] = private\_methods  
  
protected\_methods = ent.metric(['CountDeclMethodProtected']).get('CountDeclMethodProtected', 0)  
und\_protected\_results[ent\_name] = protected\_methods  
  
print("All methods : ", und\_all\_results)  
print("Default methods : ", und\_default\_results)  
print("Private methods : ", und\_private\_results)  
print("Protected methods : ", und\_protected\_results)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 test\_understand\_kinds()

تابع را روی دیتابیس ساخته شده توسط understand به نام src.und اجرا می‌کنیم و خروجی به صورت زیر است:

A picture containing text

Description automatically generated

# **مقایسه خروجی‌ها**

\* حال برای اثبات درستی توابعی که زدیم خروجی دو فایل تست را باهم مقایسه می‌کنیم:

**خروجی ما :** A picture containing text

Description automatically generated

**خروجی understand :**

A picture containing text

Description automatically generated

* همانطور که می‌بینیم تعداد‌های بدست آمده از توابعی که زدیم با خروجی understand **برابر** است.