Načini debagovanja u programskom jeziku Python

Dimitrije Sekulić, Sandra Radojević, Maja Gavrilović, Matija Pejić

Matematički fakultet, Beograd

28. april 2020.

Uvod

- Greške pri programiranju se svima dešavaju
- Debagovanje je proces nalaženja i otklanjanja grešaka u programu.
- Ono podrazumeva sledeće:
  1. Znamo kako program treba da radi
  2. Opažamo da je do baga došlo
  3. Pronalazimo bag
  4. Uklanjamo bag

Debagovanje sintaksnih grešaka

```
def student(name):
    students = {
        'Pera': '107/2016',
        'Mika': '16/2016'
        'Laza': '252/2015'
    }

    print('Index of Pera is ' + studenti[name])

student('Pera')
```

```
  File "primer.py", line 5
    'Laza': '252/2015'
          ^
SyntaxError: invalid syntax
```

Debagovanje sintaksnih grešaka

- Kada u programu postoji sintaksna greška prevodilac izbacuje izuzetak i ispisuje **poruku o grešci**. Ona sadrži:
  1. Tip greške
  2. Opis greške
  3. Traceback
- Neki izuzeci se ne mogu izbeći, takve izuzetke hvatamo korišćenjem **try** i **except** bloka
- Kada se naš program prevede, ali ne dobijamo željeni rezultat, takvu grešku nazivamo **semantičkom greškom**

Naučni pristup debagovanju

Predstavlja formalan pristup pronalaženju problema koji je zasnovan na sledećim koracima:

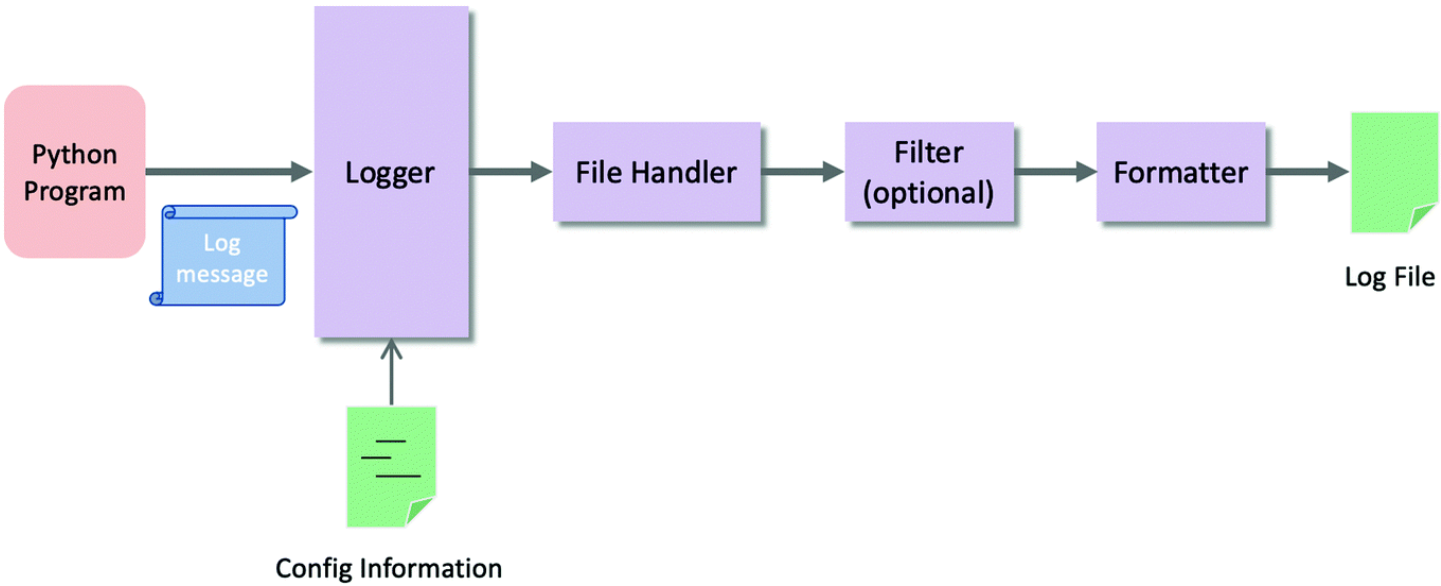
1. Posmatraj
2. Napravi hipotezu
3. Predvidi
4. Testiraj
5. Zaključi

Za efikasnu primenu, neophodno je vladati tehnikama:

- reprodukcije grešaka
- automatizacijom
- izolacijom grešaka

Debagovanje print metodom

- Jednostavna, ali moćna metoda
- Biblioteke pprint, logging daju fleksibilnost
- Neke od klasa u logging biblioteci su:
  Logger, LogRecord, Handler, Filter, Formatter



Slika: Rad sa log zapisima

PDB debager

- Interaktivni program za otklanjanje grešaka.
- Prati izvršavanje programa korak po korak i pruža pomoć pri rešavanju bagova.
- Debagovanje programa pokrećemo:
  1. iz komandne linije
  2. iz samog programa

1. Pokrećemo skript Pdb komandom
   python -m pdb imeprograma.py arg1 arg2
2. Umećemo deo koda u program na mesto odakle želimo da započnemo proces debagovanja.
   import pdb; pdb.set_trace()
   pdb.set_trace() postavlja debager za pozivajući stek okvir.

Primer

Primer prvi.py

```
my_list = [1,9,13,3,12]
new_list = list(map(lambda x: x*2,my_list))

def sub(a,b):
  print(a)
  return a-b

diff = sub(40,2)
my_list_sum = sum(my_list)
experiment = sum(new_list) / sub(diff,my_list_sum)
```

```
> prvi.py(1)<module>()
-> my_list = [1,9,13,3,12]
(Pdb) n
> prvi.py(2)<module>()
-> new_list = list(map(lambda x: x*2,my_list))
(Pdb) n
> prvi.py(3)<module>()
-> def sub(a,b):
(Pdb) n
> prvi.py(6)<module>()
-> diff = sub(40,2)
(Pdb) s
--Call--
> prvi.py(3)<module>()
-> def sub(a,b):
(Pdb) n
> prvi.py(4)sub()
->print(a)
(Pdb) n
40
>prvi.py(5)sub()
->return a-b
(Pdb) n
--Return--
>prvi.py(5)sub->38
->return a-b
```

Tamo gde želimo da istražujemo postavljamo tačke prekida.

```
import pdb; pdb.set_trace()
experiment = sum(new_list) / sub(diff,my_list_sum)
```

Ako sada program pokrenemo sa python prvi.py, prva linija za izvršavanje korišćenjem debagera biće
experiment = sum(new_list) / sub(diff,my_list_sum)
U slučaju da dodamo i argumente –m pdb, izvršavanje kreće od prve linije.

```
>prvi.py(1)<module>()
->my_list = [1,9,13,3,12]
(Pdb)c
40
>prvi.py(9)<module>()
->experiment = sum(new_list) / sub(diff,my_list_sum)
(Pdb) n
38
ZeroDivisionError: division by zero
```

Tačke prekida možemo postavljati i komandom b (break).

```
(Pdb) b 9
```

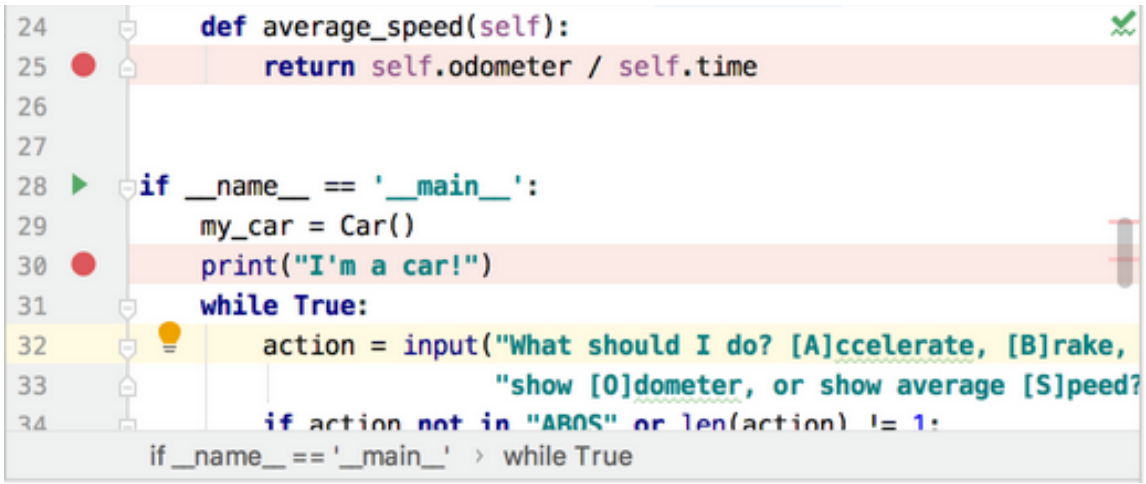
Šta je PyCharm

PyCharm je integrisano razvojno okruženje koje se koristi za programiranje u jeziku Python. Pruža analizu koda, grafički debager, integraciju sa verzijom kontrolnog sistema(git) i druge pogodnosti.
Bitni pojmovi Pycharm Debugger-a:

1. Detaljno debagovanje
2. Posmatranja
3. Inline Debugger
4. Evaluacija izraza

Tačke prekida i pokretanje Debugger-a

U okruženju PyCharm tačke prekida postavljamo klikom na levu marginu (oznaka tačke prekida je crveni kružić). Prilikom kompilacija možemo odabrati opciju Debug, nakon čega dobijamo zaseban prozor za debagovanje (Debug Tool Window)



Slika: Postavljanje tačaka prekida.

Opcije Debugger-a

Sve opcije Debugger-a se nalaze u Debug Tool Window-u.
Inline Debbuger je opcija koja nam pruža da u vidu komentara u editoru vidimo sve vrednosti promenljivih.
Evaluacija izraza je opcija koja nam omogućava da izračunamo bilo koji izraz sa trenutnim vrednostima promenljivih u kodu, kao i da dodeljujemo vrednosti promenljivama.
Posmatranja su zaseban prozor u kome se nalaze sve promenljive koje su trenutno definisane, kao i njihove vrednosti.
Detaljno debagovanje predstavlja skup opcija za iteriranje kroz kod korak po korak.

Zaključak

Sve tehnike debagovanja koje smo vam danas prikazali su jednako moćne i ne treba vršiti njihovo poređenje, već kao iskusan programer treba biti upoznat sa njima i izabrati metodu koja najviše odgovara nama i onome što radimo. Nadamo se da vam se bar nešto od ovoga dopalo i da ćete biti podstaknuti da se oprobate u nečemu što nije print metoda za debagovanje.

Hvala na pažnji!

Literatura

Kristian Rother (2017)

Pro Python Best Practices Debugging, Testing and Maintenance

Kalyani Adawadkar (2017)

Python Programming-Applications and Future

International Journal of Advance Engineering and Research Development

Python 3.8.2 documentation

`https://docs.python.org/3/`

PyCharm

`https://www.jetbrains.com/help/pycharm/`