Dagskrá í viku 12

- Miðvikudagur 26.3
  - Verkfall og lokapróf
  - Hópverkefni 2
  - Notendasögur, verkþættir og verkáætlun
  - Aflúsun (kafli 11)
  - Tímaverkefni

Verkfall og lokapróf

- Lokaprófið verður krossapróf
  - Fljótlegra að fara yfir og auðveldara að skila einkunnum sem fyrst
- Á ég að reyna að óska eftir því að halda það eins snemma og hægt er eftir verkfall?
  - Fáið einkunn fyrr og því námslán fyrr
  - Búið að ljúka því af

Hópverkefni 2

- Kennarar gefa einkunn miðað við fullgerðan kapal, ekki miðað við bónusstig.
  - Því geta skil 2C fengið 10 en 2D eingöngu 8.
- Bónusstig er hægt að fá fyrir hvað sem er og þeim mun frumlegra og/eða flóknara, þeim mun fleiri stig
- Til að fá bónusstig þarf að taka þau fram í skýrslu og rökstyðja “awesomeness”

Notendasögur, verkþættir og verkáætlun

- Notendasögur – forgangur og tími
- Verkþættir – Tími og hver vinnur verkið (upphafsstafir). Ekki horfa á tímann á notendasögunni þegar áætlað er á verkþáttinn, verkþættirnir eru nákvæmari en notendasagan svo hana má uppfæra miðað við þá.
- Verkáætlun – Kafli 4, inniheldur Notendasögur með áhengdum verkþáttum og burndown rit
  - Í upphafi ítrunar (allir verkþættir enn á notendasögum og óúthlutaðir bls 116-7).
  - Í lok ítrunar (allir verkþættir úthlutaðir, búnir og notendasögur komnar í Completed bls 134-5).

Aflúsun

- Finna **orsök** villu og **leiðréttta** síðan villuna
- Allt að helmingur tímans fer í aflúsun
- Allt að 20-faldur munur á hversu langan tíma tvo forritara tekur að finna sömu villuna
- Nýjar villur verða oft til í leiðréttingarferlinu
- Mesti tíminn fer oft í að finna og skilja villuna

Algengar forritunarvillur

- Fyrsta/síðasta stak gleymist (off by one error)
  - Dæmi: Skrifa út stök 1 til 4

```
for i in range(1, 4):
    print element[i]
```

- Ekki gengið úr skugga um að inntak sé í lagi

```
def average(lst):
    return sum(lst) / len(lst)
```

Algengar forritunarvillur

- Fyrsta/síðasta stak gleymist (off by one error)
  - Dæmi: Skrifa út stök 1 til 4

```
for i in range(1, 4):
    print element[i]
```

Úps! *i* tekur gildin 1,2 og 3
(sbr. skilgreiningu á range())

- Ekki gengið úr skugga um að inntak sé í lagi

```
def average(lst):
    return sum(lst) / len(lst)
```

Deilt með 0 ef lst er tómur

Algengar forritunarvillur

- Minnismiðlun (memory management bugs)
  - Algengt vandamál í C/C++
  - Þekkjum líka úr Python, sbr.

```
lstA=[1,2,3,4]
lstB=lstA
lstB[0]=-1
print lstA
>>>
[-1, 2, 3, 4]
```

- Óskilgreindar breytur (uninitialized variables)
  - Sjaldan vandamál í Python

Algengar forritunarvillur

- Minnismiðlun (memory management bugs)
  - Algengt vandamál í C/C++
  - Þekkjum líka úr Python, sbr.

```
lstA=[1,2,3,4]
lstB=lstA
lstB[0]=-1
print lstA
>>>
[-1, 2, 3, 4]
```

lstB og lstA vísa á sömu stökin.
Lausn: Búum til afrit af lstA með lstB=list(lstA)

- Óskilgreindar breytur (uninitialized variables)
  - Sjaldan vandamál í Python

Aflúsunarferli

1. Finna hvernig framkalla megi villuna
   - Útbúa prófunartilvik (eininga- eða kerfispróf)
2. Finna orsök villunar
3. Laga villuna
4. Prófa viðgerðina
5. Leita að svipuðum villum (t.d. deilt með 0)

Villuleit

- Endurbæta prófunartilvik
- Keyra einingapróf reglulega
- Samþætta jafnóðum
- Nota sérhæfð tól
  - "debuggera", "syntax-checkera" (t.d. **pylint**), …
- Lýsa vandamálinu fyrir samstarfsfólki
- Taka sér pásu af og til

pylint tólið

- Gefur skýrslu um ástand kóða

```
************* Module python_mandelbrot
C: 15: Line too long (108/80)
C: 61: Line too long (85/80)
C:  1: Missing docstring
C:  5: Invalid name "stdout" (should match (([A-Z_][A-Z0-9_]*)|(__.*__))$)
C: 10:Iterator: Missing docstring
C: 15:Iterator.__init__: Invalid name "y" (should match [a-z_][a-z0-9_]{2,30}$)
C: 17:Iterator.__init__: Invalid name "x" (should match [a-z_][a-z0-9_]{2,30}$)

[...] and a very long report with useful stats like :

Duplication
-----------

+-------------------------+------+---------+-----------+
|                         |now   |previous |difference |
+=========================+======+=========+===========+
|nb duplicated lines      |0     |0        |=          |
+-------------------------+------+---------+-----------+
|percent duplicated lines |0.000 |0.000    |=          |
+-------------------------+------+---------+-----------+
```

Heimild: stackoverflow

Villuleit – erfið tilfelli

- Rýna hönnun/kóða sem liggur undir grun
- Bæta nýjum kóða við í litlum skömmtum og prófa hverja uppfærslu ítarlega
  - Deila & Drottna: Breyta öllum kóða í athugasemdir og taka kóðann síðan smám saman í gagnið m.þ.a. fjarlægja athugasemdir
- Henda viðkomandi kóða og endurhanna/ forrita frá grunni

Örþrifaráð: Henda öllu forritinu

Aflúsun í Python

- Einfalt (og vinsælt)
  - Nota **print** skipunina til að sýna stöðu í forriti
- **assert** skipanir spara mikla vinnu við aflúsun

```
Line 1: def count_exact_matches(code, guess):
Line 2:     assert len(code) == 4
Line 3:     assert len(guess) == 4
            ...  # rest of implementation

>>> count_exact_matches('ABCD', 'ABCDE')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in
      count_exact_matches
AssertionError
```

Sjáum hvaða assert klikkar

Heimild: Caltech/LEAD CS

Aflúsun í Python

- logging pakkinn
  - http://docs.python.org/2/howto/logging.html#logging-basic-tutorial
  - Skrá eðlilega/óeðlilega hegðun í forriti
  - Kemur í stað print skipana

```
import logging
x=3 # Assign some value to
x
# ... do something ...
logging.warning('x=%d', x)

WARNING:root:x=3
```

Birtist í "console" glugga

Úttak fer í skrá

```
import logging
x=3 # Assign some value to x
# ... do something ...
logging.basicConfig(filename='example.log',
                    level=logging.DEBUG)
logging.debug('x=%d', x)
```

Alvarleikastig: debug, info, warning (sjálfvalið), error, critical

Aflúsunartól (debugger)

- Innbyggð í V. Studio, Eclipse, Matlab, IDLE, ...
- Skilgreinum rofstaði (e. breakpoint) í forriti
  - Stöðva keyrslu í tiltekinni línu eða falli
  - Stöðva keyrslu þegar tiltekin skilyrði eru uppfyllt
  - Hægt að rekja sig áfram línu fyrir línu
  - Hægt að skoða innihald breyta og gagnagrinda
  - Rekja sig gegnum stef, hoppa yfir stef

Aflúsunartól í IDLE

- Kóði sem virkar ekki sem skyldi

```
import random
number1 = random.randint(1, 10)
number2 = random.randint(1, 10)
print('What is ' + str(number1) + ' + ' + str(number2) + '?')
answer = raw_input()
if answer == number1 + number2:
    print('Correct!')
else:
    print('Nope! The answer is ' + str(number1 + number2))
```

- Úttak

```
What is 2 + 4?
6
Nope! The answer is 6
```

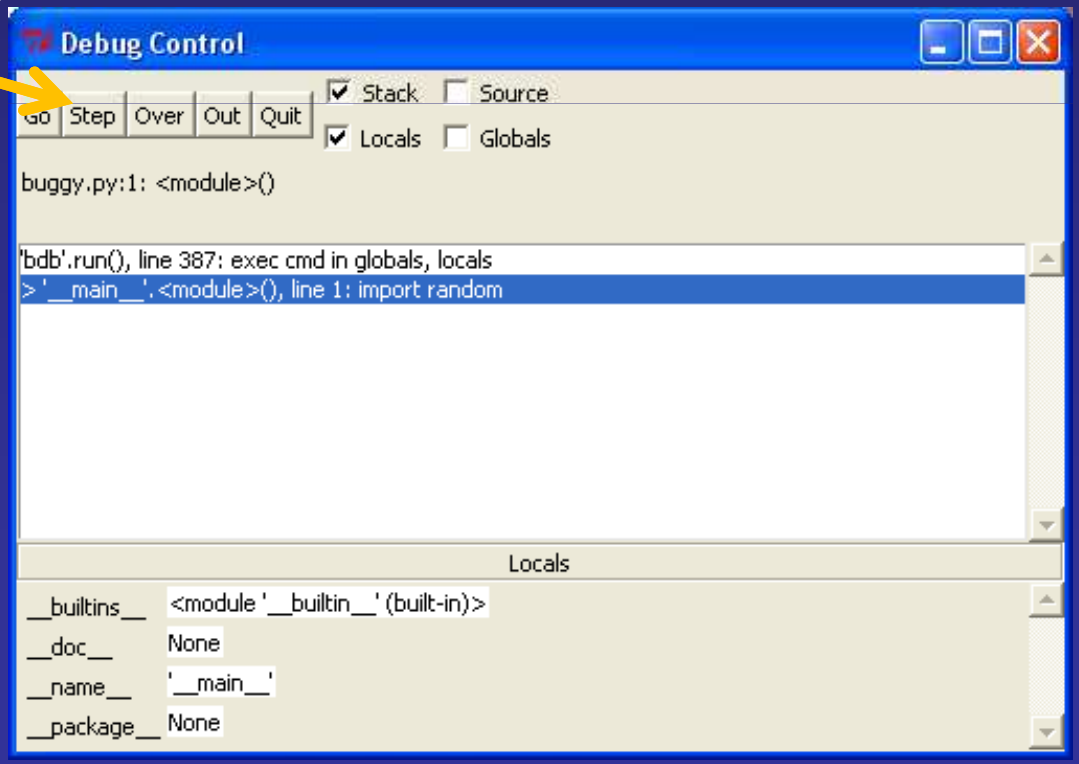
ATH Mörg fullkomnari tól í boði, t.d. Winpdb (cross-platform) og pudb

Heimild: inventwithpython.com

Aflúsunartól í IDLE

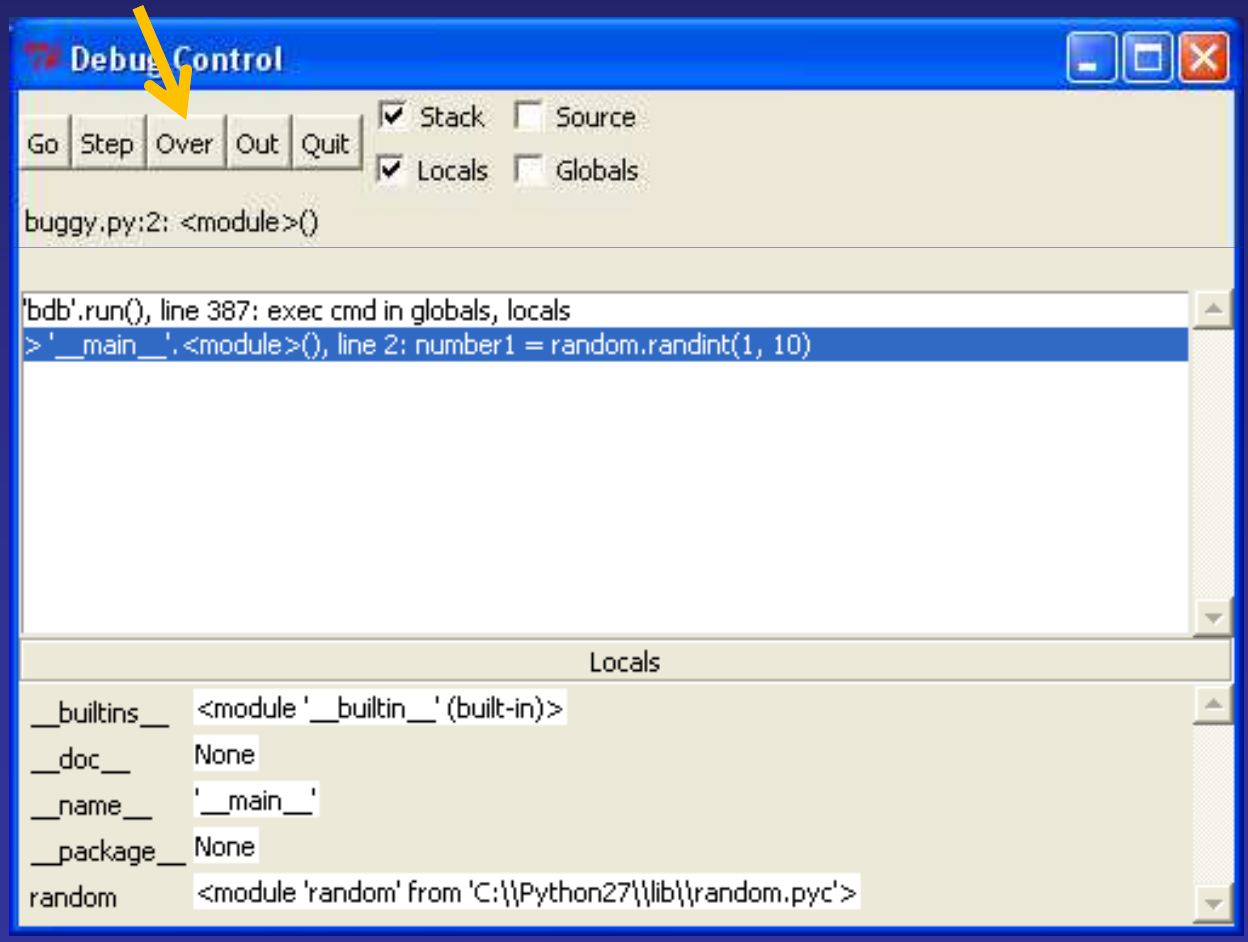
- Velja Debug / Debugger í Python skelinni
- Keyra forrit (t.d. með F5 – Run module)

Keyra eina línu í einu, ef um fallskall er að ræða þá er farið inn í fallið

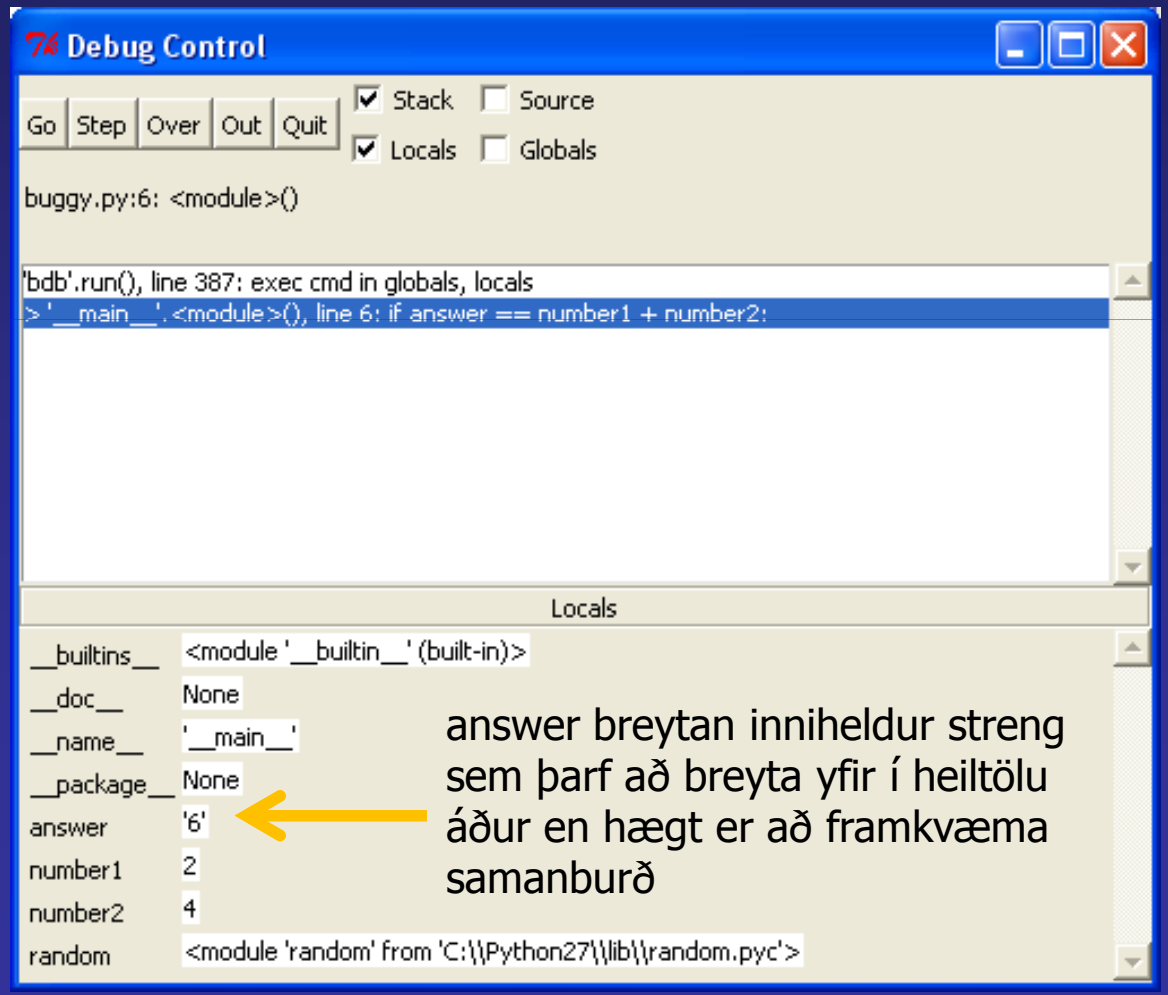


Aflúsunartól í IDLE

Keyra eina línu í einu,
ekki er farið inn í föll/stef



Aflúsunartól í IDLE



Rofstaðir

(breakpoints)

- Hægri smella á línu og velja 'Set breakpoint'

```
import random
number1 = random.randint(1, 10)
number2 = random.randint(1, 10)
print('What is ' + str(number1) + ' + ' + str(number2) + '?')
answer = raw_input()
if answer == number1 + number2:
    print('Correct!')
else:
    print('Nope! The answer is ' + str(number1 + number2))
```

- 'Go' keyrir forrit þar til komið er að línunni

Leiðréttingar á villum

- Skilja kóðann, ekki bara vandamálið
- Sannreyna greininguna
  - til að lenda ekki á villigötum
- Slappa aðeins af ... og uppfæra svo
  - fá samstarfsmann til að horfa yfir öxlina á sér
- Geyma upprunalega kóðann ← Gerist sjálfkrafa með útgáfustýringu

Leiðréttingar á villum

- Gildar ástæður liggi að baki breytingum
  - Ekki breyta t.d. $i < n$ í $i \leq n$ án góðrar ástæðu
- Ganga úr skugga um að viðgerðin hafi tekist
- Búa til einingapróf sem framkallar villuna
  - Þurfum a.m.k. ekki að leita sömu villu uppi aftur
- Gera eina breytingu í einu

Leiðréttingar á villum

- Laga vandamálið, ekki einkennin
  - Dæmi: G.r.f. að útkoma úr eftirfarandi kóða sé $3.45 of lág þegar client tekur gildið 45

```
for ( claimNumber = 0; claimNumber < numClaims[ client ]; claimNumber++ ) {
   sum[ client ] = sum[ client ] + claimAmount[ claimNumber ];
}
```

- „Viðgerð" sem gerir illt verra

```
for ( claimNumber = 0; claimNumber < numClaims[ client ]; claimNumber++ ) {
   sum[ client ] = sum[ client ] + claimAmount[ claimNumber ];
}
if ( client == 45 ) {
   sum[ 45 ] = sum[ 45 ] + 3.45;
}
```

Sýnidæmi úr kafla 11 í HFSD

- Síðbúin krafa viðskiptavinar um að nota utanaðkomandi pöntunarkerfi (mmeals)
  - kóðarýni á mmeals leiðir í ljós ýmisa annmarka
- Höfum 3 notendasögur sem nota mmeals
  - forritið frýs þegar kallað er á mmeals
- Viðskiptavinur látinn vita að tímaplön séu í uppnámi

Sýnidæmi úr kafla 11 í HFSD

- Skref til að leysa vandann
  - setja mmeals undir útgáfustýringu
  - fá mmeals Java kóðann til að þýða
  - skrifa einingapróf m.t.t. notkunar okkar á mmeals
  - samþætta einingapróf og útgáfustýringu (vika 5)
  - tilkynna viðskiptavini að endurskoða þurfi tímamat

Sýnidæmi úr kafla 11 í HFSD

- Skref til að leysa vandann – frh.
  - lögum eingöngu villur sem snúa að þeirri virkni mmeals sem við þurfum á að halda
  - útbúa einingapróf sem prófa þessa virkni
  - Dæmi: 30% prófanna falla
- Velja nokkur próf sem falla af handahófi
  - reyna að lagfæra viðkomandi villur í mmeals
  - fáum mat á hversu langan tíma tekur að laga allar villur í mmeals sem skipta okkur máli

Tímaverkefni 12

1. Í hvaða tilfellum myndum við commenta allan kóðann út og setja eitt fall í einu inn?
2. Hvernig hefðum við getað leyst vandamálið með floating point error í einstaklingsverkefni 1 án þess að bæta 0.00001 við?
3. Hvað er aflúsun?
4. Er aflúsun það sama og prófun?

Ítarefni

- Aflúsun í Python
  - http://courses.cms.caltech.edu/lead/lectures/lecture09.pdf
  - http://www.cs.washington.edu/education/courses/cse140/13wi/lectures/08-debugging.pdf