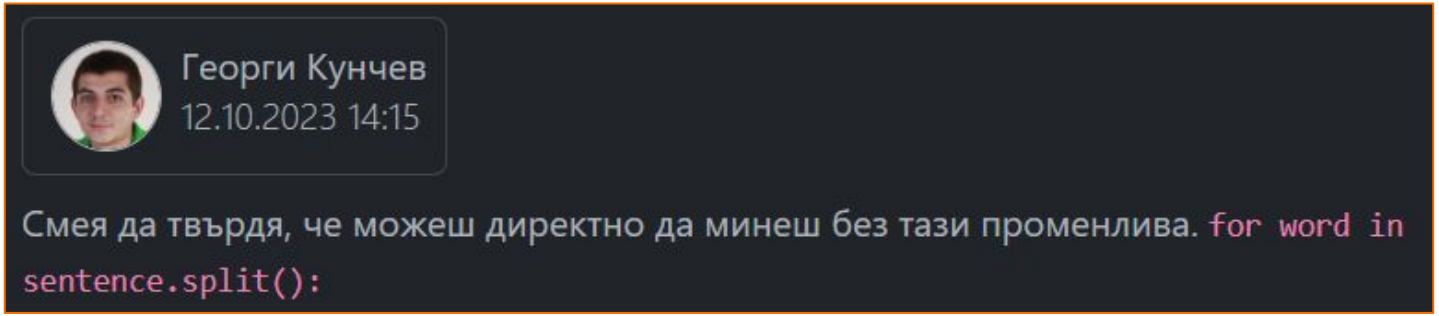
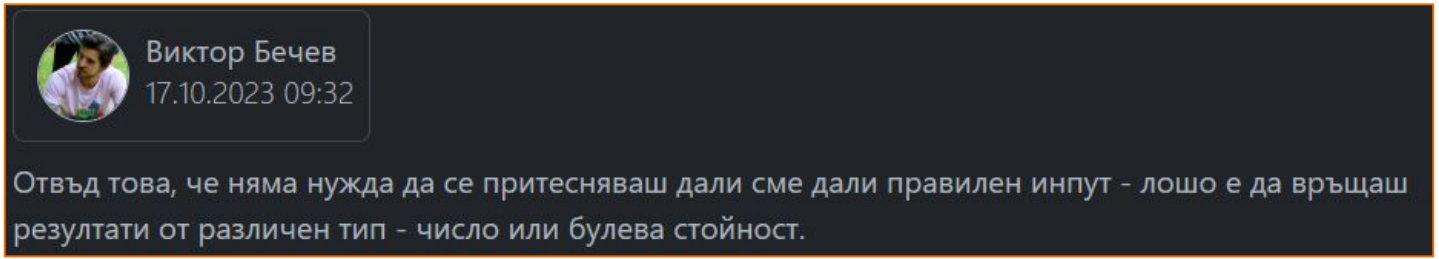
05. Домашни дискусии

19 октомври 2023

Не такива домашни дискусии

А такива домашни дискусии





Но първо един въпрос

```
def spam():
    return spam

type(spam) # ?
```

```
<class 'function'>
```

...и още един

```
def spam():
    return spam

type(spam()) # ?
```

```
<class 'function'>
```

...и още един

```
def spam():
    return spam

spam()()()()() is spam # ?
```

```
True
```

...и още един

```
def spamifize(f):
    def spamifized(*args, **kwargs):
        return f(*args, **kwargs)
    return spamified

def spam():
    return spam

spamified_spam = spamifize(spam)

spam() is spamified_spam() # ?
```

```
True
```

...и още един

```
spamifize = lambda x: lambda: x

def spam():
    return spam

spamifized_spam = spamifize(spam)

spam() is spamifized_spam() #?


True
```

Обратно по темата - Основни (положителни) изводи

- Вече знаете как да качвате домашни си
- Имаме около 90% активност в предаване на домашни
- Качвате ги сравнително рано и имаме време да дадем обратна връзка
- Понаучихте се на PEP8 от коментарите ни
- Понаучихте се да търсите чист и кратък код

И все пак има неща, които искаме да обсъдим

- синтактични излишъци
- алгоритмични излишъци
- проблеми с форматиране на кода
- пропуски при проверка на качеството
- оптимизации на кода
- лоши практики
- бонус точки

Но това не е всичко

Ще поговорим за

- писане на тестване
- (не)четим, но готин код

Синтактични излишъци (1)

Кое е излишното?

```
return word[0:size]


return word[:size]
```

Синтактични излишъци (2)

Кое е излишното?

```
return word[size:len(word)]
```

```
return word[size:]
```

Синтактични излишъци (3)

Кое е излишното?

```
return word[-1::-1]
```

```
return word[::-1]
```

Синтактични излишъци (4)

Кое е излишното?

```
splitted = (beginning(word), middle(word), end(word))
result.append(splitted)


result.append(beginning(word), middle(word), end(word))
```

Синтактични излишъци - защо да ми пука?

- По-четим код
- Спестяване на излишни операции - т.е. на време и място
- Спестяване на време за писане
- По-дълбоко разбиране на езика
- PEP8
- Останалите програмисти ви обичат
- Ние предупредихме още в нулевата лекция: Grammar Nazis

Алгоритмични излишъци (1)

Кое е излишното?

```
for i in range(len(words)):
    beg_word = beginning(words[i])


for word in words:
    beg_word = beginning(word)
```

Алгоритмични излишъци (2)

Кое е излишното?

```
if length % 3 == 0:
    return word[:size]
elif length % 3 == 1:
    return word[:size]
else:
    return word[:size + 1]
```

```
if length % 3 in (0, 1):
    return word[:size]
else:
    return word[:size + 1]
```

Алгоритмични излишъци (3)

Кое е излишното?

```
word = ''
words = []
for letter in sentence:
    if letter != ' ':
        word += letter
    else:
        words.append(word)
        word = ''


words = sentence.split()
```

Алгоритмични излишъци (4)

Кое е излишното?

```
size = int(word_length / 3)
```

```
size = word_length // 3
```

Алгоритмични излишъци - защо да ми пука?

- По-четим код
- Спестяване на излишни операции - т.е. на време и място
- Спестяване на време за писане
- По-дълбоко разбиране на езика
- Останалите програмисти ви обичат

Проблеми с форматиране на кода (1)

Къде е проблемът?

```
return (beginning(word),middle(word),end(word))
```

```
return beginning(word), middle(word), end(word)
```

Проблеми с форматиране на кода (2)

Къде е проблемът?

```
word_length=len(word)


word_length = len(word)
```

Проблеми с форматиране на кода (3)

Къде е проблемът?

```
def separate_word(word):
    return (beginning(word), middle(word), end(word))
def split_sentence(sentence):
    return list(map(separate_word, sentence.split()))
```

```
def separate_word(word):
    return (beginning(word), middle(word), end(word))

def split_sentence(sentence):
    return list(map(separate_word, sentence.split()))
```

Проблеми с форматиране на кода (4)

Къде е проблемът?

```
wordLength = len(word)


word_length = len(word)
```

Проблеми с форматиране на кода (5)

Къде е проблемът?

```
# Return a split version of a sentence
def split_sentence(sentence):
    return sentence.split()



def split_sentence(sentence):
    """Return a split version of a sentence."""
    return sentence.split()
```

Проблеми с форматиране на кода (6)

Къде е проблемът?

```
return word[start: end]
```

```
return word[start:end]
```

However, in a slice the colon acts like a binary operator, and should have equal amounts on either side (treating it as the operator with the lowest priority). In an extended slice, both colons must have the same amount of spacing applied. Exception: when a slice parameter is omitted, the space is omitted:

Проблеми с форматиране на кода (7)

Къде е проблемът?

```
if (word_length % 3 == 0):
    return word[:size]
```

```
if word_length % 3 == 0:
    return word[:size]
```

Проблеми с форматиране на кода - защо да ми пука?

- По-четим код
- По-дълбоко разбиране на езика
- PEP8
- Останалите програмисти ви обичат

Концептуални проблеми

Къде е проблемът?

```
class HelloWorld {
    public static void main(String args[])
    {
        System.out.println("Hello, World!");
    }
}
```

```
print("Hello, World!")
```

Пропуски при проверка на качеството (1)

Къде е проблемът?

```
def split_sentence(sentence):
    result = []
    for i in sentence.split():
        triple = (beginning(i), middle(i), end(i))
        result.append(triple)
    print(result)


def split_sentence(sentence):
    result = []
    for i in sentence.split():
        triple = (beginning(i), middle(i), end(i))
        result.append(triple)
    return result
```

Пропуски при проверка на качеството (2)

Къде е проблемът?

```
def split_sentence(sentence):
    return [(beginning(i), middle(i), end(i)) for i in sentence.split()]

test = input()
print(split_sentence(test))
```

```
def split_sentence(sentence):
    return [(beginning(i), middle(i), end(i)) for i in sentence.split()]

# test = input()
# print(split_sentence(test))
```

Пропуски при проверка на качеството (3)

Къде е проблемът?

```
if type(element) is int or float:
    result.append(element * -1)


if type(element) is int or type(element) is float:
    result.append(element * -1)

# или

if type(element) in (int, float):
    result.append(element * -1)
```

Пропуски при проверка на качеството - защо да ми пука?

Класиране

1	 Dr. Evil	15
2	 Number Two	14

Оптимизации на кода (1)

Къде е проблемът?

```
if length % 3 == 0 or length % 3 == 1:
    return word[size:]
```

```
if length % 3 in (0, 1):
    return word[size:]
```

Оптимизации на кода (2)

Къде е проблемът?

```
def middle(word):
    length = len(word)
    if length % 3 == 0:
        return word[length // 3:2 * (length // 3)]
    if length % 3 == 1:
        return word[length // 3:2 * (length // 3) + 1]
    if length % 3 == 2:
        return word[length // 3 + 1:2 * (length // 3) + 1]
```

```
def middle(word):
    end_size = round(len(word) / 3)
    return word[end_size : len(word)-end_size]
```

Оптимизации на кода - защо да ми пука?

- По-четим код
- Спестяване на излишни операции - т.е. на време и място
- Спестяване на време за писане
- По-дълбоко разбиране на езика
- Останалите програмисти ви обичат

Лоши практики (1)

Къде е проблемът?

```
# This is a comment, explaining the next few lines
# but I have no space to write it in a single line
# so I need multiple lines.
```

```
"""
This is a comment, explaining the next few lines
but I have no space to write it in a single line
so I need multiple lines.
"""
```

Лоши практики (2)

Къде е проблемът?

```
def middle(word):
    if type(word) is not str:
        print(f"{word} is not a word :/ input a string.")
        return None
    <code>


def middle(word):
    # Design by contract
    <code>
```

Лоши практики (3)

Къде е проблемът?

```
def middle(str):
    size = round(length(word) / 3)
    return str[:size]


def middle(word):
    size = round(length(word) / 3)
    return word[:size]
```

Лоши практики (4)

Къде е проблемът?

```
l = len(word)


word_length = len(word)
```

Лоши практики (4++)

Къде е проблемът?

```
def split_sentence(sentence):
    res = []
    for word in sentence.split():
        res.append(beginning(word), middle(word), end(word))
    return res


def split_sentence(sentence):
    words = []
    for word in sentence.split():
        words.append(beginning(word), middle(word), end(word))
    return words
```

Лоши практики (6)

Къде е проблемът?

```
def middle(input):
    size = round(length(input) / 3)
    return input[:size]


def middle(word):
    size = round(length(word) / 3)
    return word[:size]
```

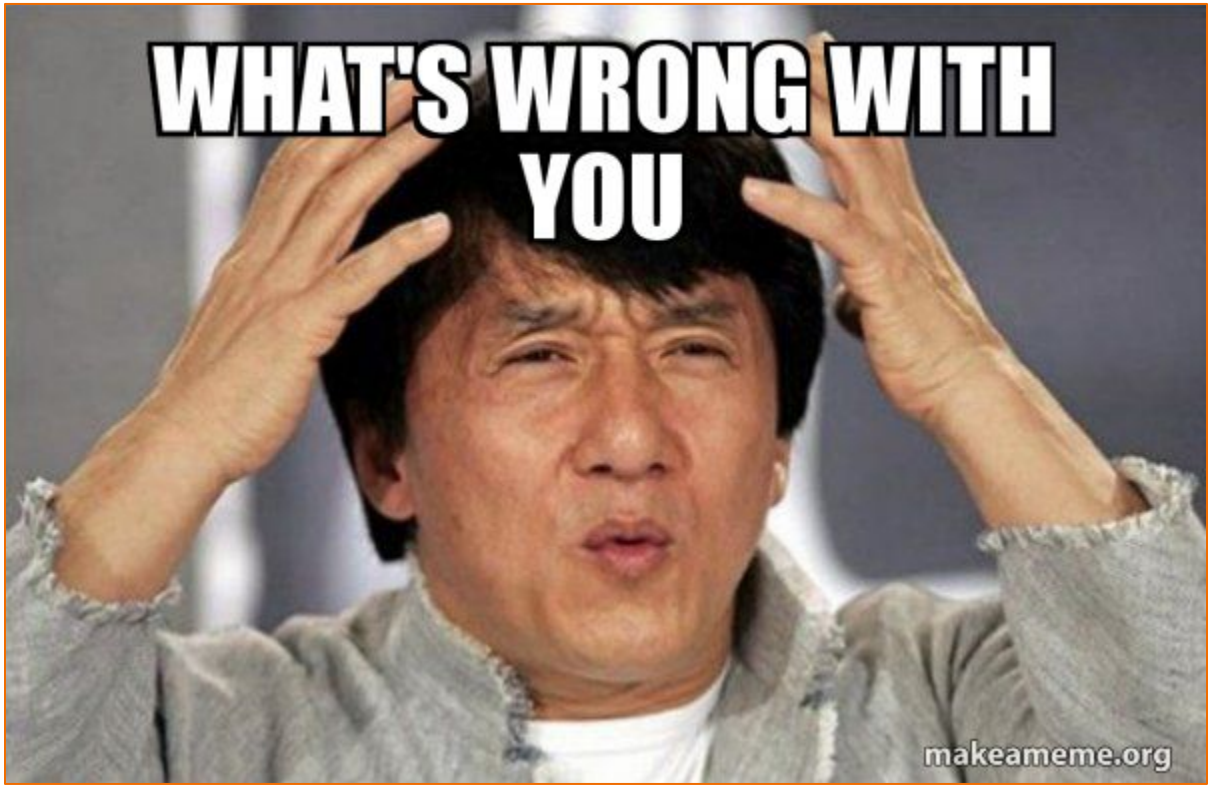
Лоши практики (6++)

Къде е проблемът?

```
#print(f"On Ith: {beginning("Good")}")
#print(f"J Sm: {middle("Good")}")
#print(f"O E: {end("Good")}")
#print()
#print(f"On Ith: {beginning("Evening!")}")
#print(f"J Sm: {middle("Evening!")}")
#print(f"O E: {end("Evening!")}")
```

Чистете си, моля!

Лоши практики - защо да ми пука?



Лоши практики (6)

Къде е проблемът?

```
def middle(input):
    size = round(length(input) / 3)
    return input[:size]
```

Проблемът е, че този слайд вече го видяхме.

Бонус точки

За тези от вас, за които сме преценили, че са ни впечатлили, има бонус точка.

И ние сме хора.

И ний сме дали нещо на света,
на вси Питонджии 1-liners да четат...

```
beginning, middle, end, split_sentence = (lambda w: w[:round(len(w) / 3)],
lambda w: w[round(len(w) / 3):len(w) - round(len(w) / 3)], lambda w: w[len(w) -
round(len(w) / 3):], lambda s: [(beginning(w), middle(w), end(w)) for w in
s.split()])
```

...и да реват

```
no_it_isnt = lambda x: [type(y)(str((y + (-2*y) + (isinstance(y, bool) or
type(y)())) or '')[::(-1)**isinstance(y, str)] or 0*y) for y in x][::-1]
```

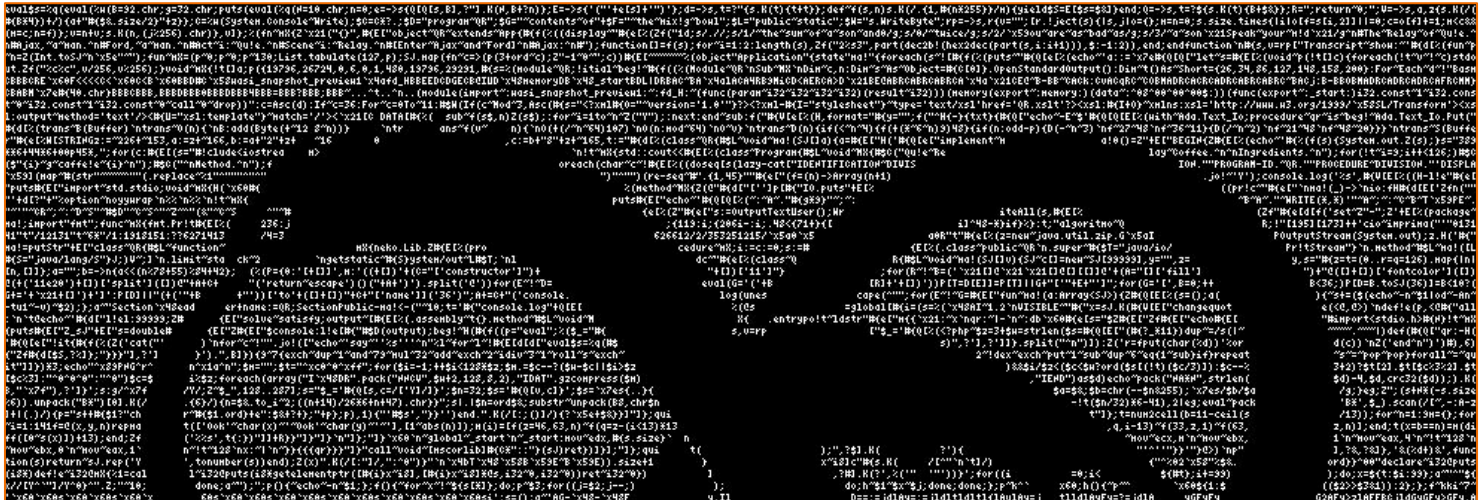
И миналата година ревяха

```
_, Card, _, Deck, _, Player, _, Game, Belot, Poker = (Card := type('Card', (), {'__init__':
lambda _, s, f: (setattr(_, 's', s), setattr(_, 'f', f))[0], 'get_suit': lambda _: _.s,
'get_face': lambda _: _.f})), Card, (Deck := type('Deck', (), {'__init__': lambda _,
ff=['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']: setattr(_, 'd',
[Card(s, f) for s in ['clubs', 'diamonds', 'hearts', 'spades'] for f in ['2', '3', '4', '5',
'6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A'] if f in ff]),'get_cards': lambda _: _.d,
'shuffle': lambda _: shuffle(_.d),'cut': lambda _: setattr(_, 'd', _.d[(cut := randint(1,
len(_.d) - 1)):] + _.d[:cut])})), Deck, (Player := type('Player', (), {'__init__': lambda _:
setattr(_, 'c', []), 'get_cards': lambda _: _.c, 'tc': lambda _, cs: _.c.extend(cs), 'rc':
lambda _: [_.c.pop() for __ in range(len(_.c))]})), Player, (Game := type('Game', (),
{'__init__': lambda _, n, dd, di, df=None: (setattr(_, 'ps', [Player() for _ in range(n)]),
setattr(_, 'd', Deck(df) if df else Deck()), setattr(_, 'dd', dd), setattr(_, 'di', di))[0],
'get_players': lambda _: _.ps, 'prepare_deck': lambda _: setattr(_.d, 'd', _.d.d +
sum([p.rc() for p in _.ps], [])), 'deal': lambda _, fp: [p.tc([_.d.d.pop() for __ in
range(di)]) for o in (_.ps[::1 if _.dd == 'ltr' else -1],) for f in (o.index(fp),) for di in
_.di for p in o[f:] + o[:f]], 'get_deck': lambda _: _.d})), Game, type('Belot', (Game,),
{'__init__': lambda _: Game.__init__(_, 4, 'ltr', (2, 3, 3), ['7', '8', '9', '10', 'J', 'Q',
'K', 'A'])}), type('Poker', (Game,), {'__init__': lambda _: Game.__init__(_, 9, 'rtl',
(1,)*5)})
```

Докато сме на темата обфускация

```
#define    F getchar())
 #define H(z)*n++=z;
      #include        <setjmp.h>
    #define v main(0,0,0
          #define Z while(
                                                  #define _ if(
#define o(d){S=63,u[1]=0,1[d]=6^e,q=1e4>v,0),1[d]=0,u[1]=e^6,S=b,q)
#define I(H,n) { _ r=1[x=H],!r|(r^e)<-1){ _ j=u[1],-7==r|6==r\
){ n; e==e; return 1e5-443*f; } u[1]=0,t=j+1,i=j-1; _!i&89<\
x)i=j,t=6; _-1==t&30>x)t=j,i=-7; Z++ict){ d =0; S&= 63; \
a=((j^e)!=1?6!=(j^e)?O[32+x/10]-O[u/10+32]-q:(S|=6!=j?8\
:1,2==u-x)*9+9*(x-u==2):(d=1==j?x-u:u-x)/8+!(!((x-u)%\
10)|r)*99+(j==1?90<x:29>x)*(9*O[28+i]-288))+O[r+28\
]*9-288+O[x%10+33]-f-O[33+u%10]; x[1]=i; S|=(21=\
=u|21==x)*2+(u==28|28==x)*4+(91==u|x==91)*16+32\
*(u==98|x==98)+(20==d)*64*x; a-=k>f?main(a,f+1\
,M,k):0; _ i==c&u==h&!f&N&a>-1e4&x==y)longjm\
p(z,1); S=b; _!N|f&&(a>M||!f&a==M&&1&rand()\
)){ _!f){ _ k){ c=i; h=u; y=x; } } else _ \
L-a<N){ n; e==e; u[1]=j; x[1]=r; return\
 a; } M=a; } } x[1]=r; u[1]=j; n; } }
typedef int G; char J [ 78 ], O [ ]
=   "HRQAMS#-smaqrh[UTZYTU[|TBA("
"S#(ABT|ba`gg`ab8>GK[_`fFDZXEYR"          "L\t####"
"###8A#8#C#F#E#O#K\t\3Z1v#tjm"           "\3J#tjm\3Pwb"
"ofnbwf\3Joofdbo\3)&`&`.&`&`"            "#+&g*\t"; G y,
c,h,e,S,*s,1[149]; jmp_buf z             ; G main(G L,G f,
G N,G k){ G u=99,p,q,r,j,i,x              ,t,a,b=S,d,M=-1e9
; char *n; if( *1){ e==e; Z        u >21){ q= 1[--u]^e;
_!-- q){ _!1[p=e?u-10:u+10]){   I(p,)_ e?u>80   & !1[p
-=10]:u<39&!1[p+=10])I(p,)} _ 1[p=e?u-11:9+u]   )I(p,)
else _ u-1==S>>6){ 1[u-1]=0; I(p,1[u-1]=-2^e);  } _ 1[
p=e?u-9:11+u])I(p,)else _ S>>6==1+u){ 1[1+u]=0; I(p,1
[1+u]=e^-2); } } _!--q){ n=O+41; Z++n<SO+O)I(u+80-*n,
)} _ O<q&4>q){  n=q==2?53+O:O+49; Z++n<O+(q!=1)*4+54
){ p=u; do I(p-=*n-80,)Z!p[1]); } } _ 4==q){ n=49+O
 ; Z++n<O+58)I(u-*n+80,)_ e&!(S&24)|!e&!(S&3)   &&
 !1[u-2]&!1[u-1]&!1[u-3]&&o(u)&&o(u-1)){ 1[u-1]=4
  ^e; 1[u-4]=0; I(u-2,1[u-1]=0; 1[u-4]=e^4); } _
  e&!(S&40)|!e&!(S&5)  &&!1[u+1]&!1[2+u]&&o(u)&&
   o(1+u)){ 1[u+1]=e^4; 1[3+u]=0;   I(u+2,1[1+u
   ]=0; 1[u+3]=4^e); } } } e==e;   return M; }
    Z h<130){1[h]=-(21>h|98<h|2       >(h+1 )%
    10); O[h++]^=3; } n=O +14;       s=20+1; Z
     ++s<29+1){ 10[s]=1; 70[s]=~    ( * s = *
      n++ -+84); 60 [ s] =-2; } Z  n=J){ puts
       (58+O); u=19; Z++u<100){ H(32)_!( u%10
       ))H(32)H(O[7+1[u]])_(9+u)%10>7){ H(58
        -u/10)H(32)_ u&1)puts(n=J); } } puts
         (O+58); _-1e4 >v , 1)){ e==e; puts
          (O+(v,0)> 1e4?e?90:82:96)); break
           ; } _ 1<i&e) { d=v,2+L); printf
            (O+114,h%10+64,58-h/10,y%10+64
             ,58 -y/10,d); } else{ putchar
              (62+e); h= (95 & F-44; c=1[h
               +=(56-F *10]; y=(95&F-44; y
                 +=(56-F*10; z 10!=(u=(95
                  &F)){ c=5; Z--c>1&&u!=c
                   [O]); c=e^c-7; } } _!
                    setjmp(z)){ v+1,1);
                        puts(   106+
                         O); }   } z
                          10!=
                           F; }
```

Докато сме на темата обфускация



05 ½. Автоматизирано тестване

19 октомври 2023

За какво няма да си говорим

- За quality assurance
- За acceptance testing
- За тестове тип “бенчмарк”
- За сомелиерство на ракия

Митът

- Проектът идва с готово, подробно задание.
- Прави се дизайн.
- С него работата се разбива на малки задачи.
- Те се извършват последователно.
- За всяка от тях пишете кода.
- Разцъквам го малко - няколко print-а, малко пробване в main метода/функцията и толкова.
- Profit.

Реалността

- Това в началото с ясните изисквания и дизайн е утопия.
- Писането на код е сложна задача - допускат се грешки.
- Програмистите са хора - допускат грешки.
- Промяната на модул в единия край на системата като нищо може да счупи модул в другия край на системата.
- Идва по-добра идея за реализация на кода.
- Често се налага един код да се преработва.

Как да автоматизираме

- За всичко съмнително ще пишем **сценарий**, който да "цъка".
- Всеки сценарий ще изпълнява кода и ще прави няколко **твърдения** за резултатите.
- Сценариите ще бъдат обединени в **групи**.
- Пускате всички тестове с едно бутонче.
- Резултатът е "Всичко мина успешно" или "Твърдения X, Y и Z в сценарии A, B и C се оказаха неверни".

Например

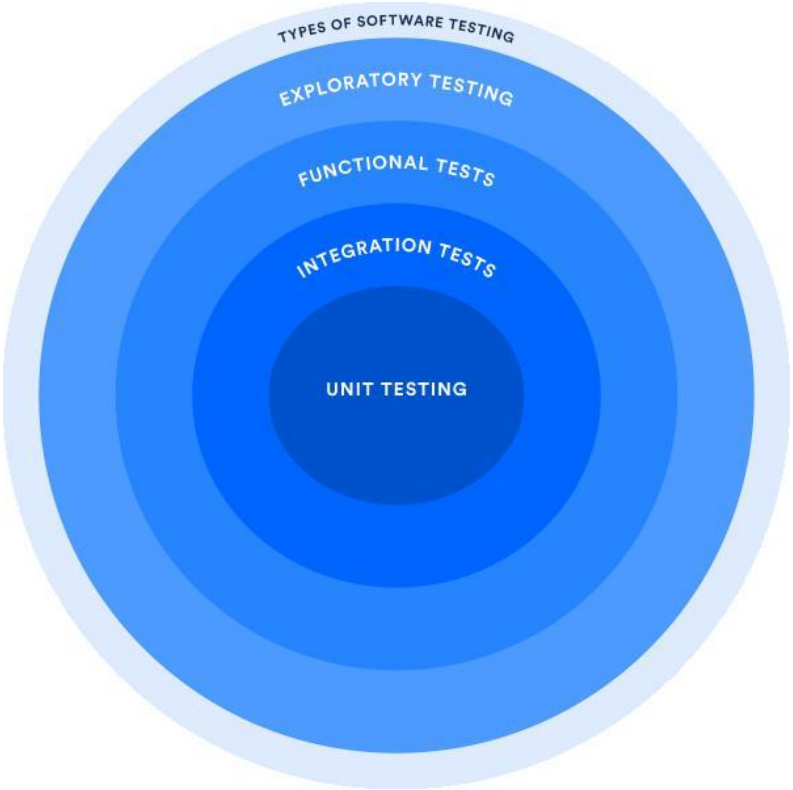
```
.F....F.....
======================================================================
FAIL: test_mixed_sentence (test.TestSentence)
Test with mixed remainder input.
----------------------------------------------------------------------
Traceback (most recent call last):
File "/tmp/test.py", line 75, in test_mixed_sentence
self.assertEqual(split_sentence('Здравейте момчета къде ми е отвертката'),
AssertionError: Lists differ: [('Зд[49 chars]', 'е'), ('м', '', 'и'), ('', '', 'е'), ('отв', 'ертк', 'ата')] != [('Зд[49 chars]', 'е'), ('м', '', 'и'), ('', 'е', ''), ('отв', 'ертк', 'ата')]

First differing element 4:
('', '', 'е')
('', 'е', '')
```

За какво ни помагат тестовете

- Откриват грешки по-рано.
- Позволяват ни уверено да правим промени в системата.
- Дават сигурност на клиенти, шефове и програмисти.
- Представляват пример как се работи с кода.
- Служат като документация и спецификация.

Типове тестове

- При преминаване към по-външните кръгове на *(ада)* тестването започваме да работим в условия, по-близки до реалните
- **Unit** тестовете трябва да се концентрират върху изолирана *(малка)* част от функционалността и да верифицират правилната ѝ работа



Имаме 4 фази на тестване

- Setup - конфигурираме “контекста” на тестването
- Execute - реалното изпълнение на тестваното
- Verify - оценяваме дали крайният резултат е очакваният
- Teardown - връщаме “контекста”, променен по време на “Setup” фазата в начално състояние

По-конкретно?

1. import unittest
2. unittest.TestCase
3. setUp / tearDown
4. assert*
5. Всеки тест трябва да е напълно независим от останалите
6. **unittest.main()** = магия

```
import unittest

class TestStringMethods(unittest.TestCase):

    def setUp(self):
        self.value = 'hmmmm'
        print(self.value)

    def tearDown(self):
        del self.value

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])

if __name__ == '__main__':
    unittest.main()
```

Как се пишат тестове?

- **import unittest**
- **unittest.TestCase**
- assertTrue(expression)
- assertFalse(expression)
- assertEqual(expected, actual)
- assertNotEqual(expected, actual)
- assertIs(expected, actual)
- assertIsNot(expected, actual)
- assertIsNone(expression)
- assertIsNotNone(expression)
- assertIn(element, collection)
- assertNotIn(element, collection)
- assertIsInstance(object, type)
- assertNotIsInstance(object, type)
- И много други...

Само unittest?

- Добре де, има и друг вариант...
- **import pytest**
- Има разлики между двете
- Няма да задълбаваме в разликите, и двете са мощни и двете се използват масово

Кога да тестваме?

A. Никога
B. След като напишем функционалността
C. Преди да напишем функционалността
D. Когато ни е скучно и нямаме какво да правим

Според Test-Driven Development парадигмата, отговорът е:

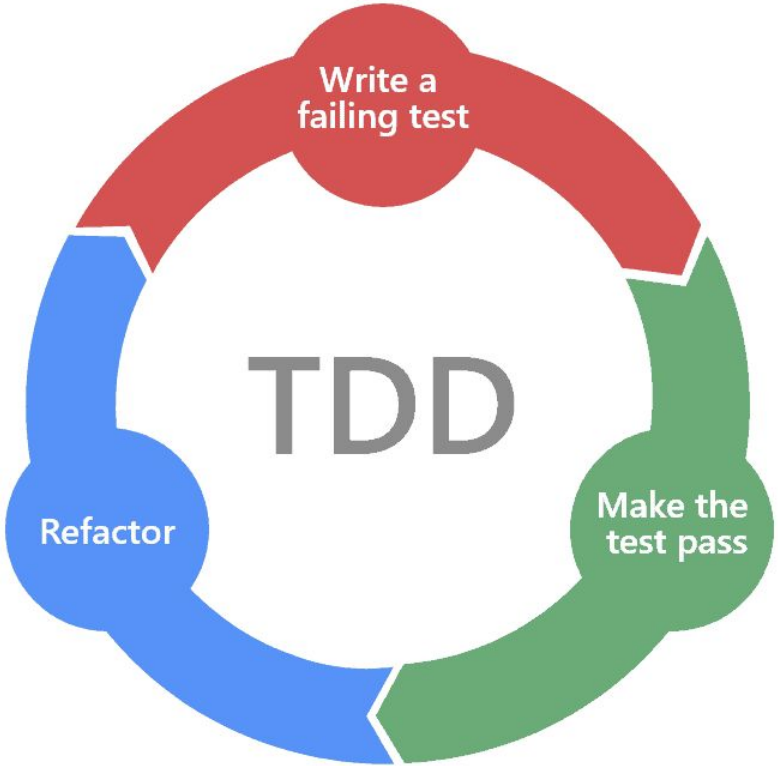
C. Преди да напишем функционалността

В зависимост от това колко точно скучно ни е, може и D.

TDD

- Test-Driven Development is not about testing.
- Подход за писане на код.
- Дизайнът е базиран върху обратна връзка, не гадаене.
- Спестява излишен код - пишете само каквото ви трябва.
- Спестява излишна функционалност.

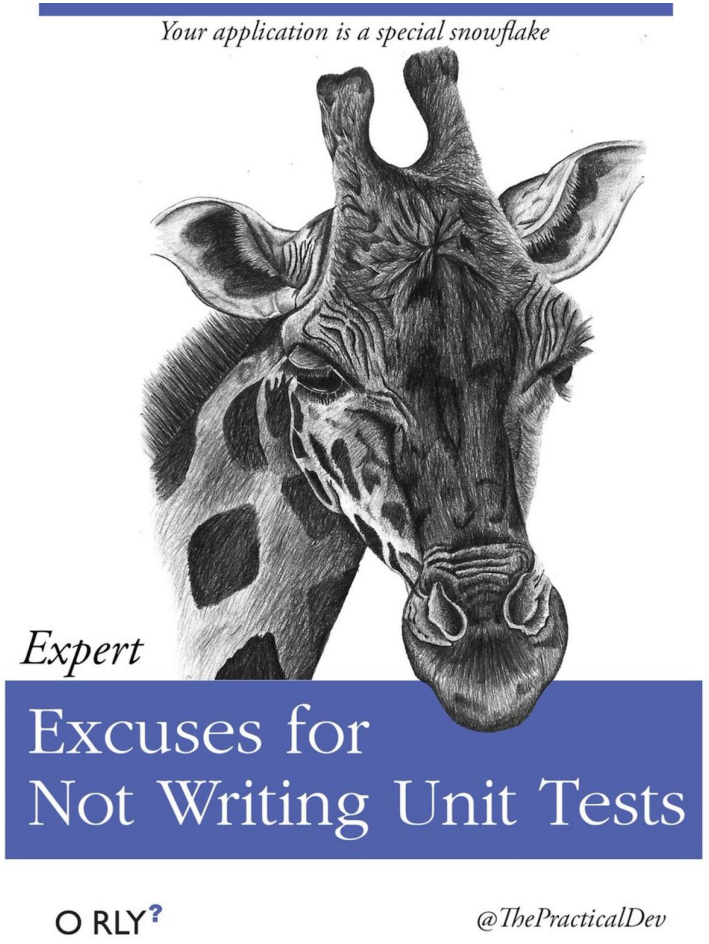
TDD (наглядно)



Добри практики при писане на тестове

- Пишете тестове за всичко, което може да се счупи.
- Не пишете тестове, които **винаги** ще минават, дори и да счупите целия codebase.
- Добър начин да си мислите за тестовете е като requirements.
- Не тествайте елементарен код.
- Успехът на тестовете не трябва да зависи от реда им.
- Тествайте гранични случаи!
- Не правете тестовете зависими един от друг.

Welcome to the real world...



TDD (още по-нагледно)

Полиморфичен негативизъм

Въпроси?