Documentaţie - Testare in Python

Barbarasa Maria-Cristina-311

Oproiu Matei-311

1.Testarea funcţională clasei ModuloGCD

(a)Partiţionare de echivalenta

1.Domeniul de intrări: Pentru clasa compusa avem 3 intrări:

- a- valoare naturală strict pozitivă deci se disting două clase de echivalența:
  A_1={ a | a număr natural strict pozitiv}
  A_2={ a | a nu este număr natural strict pozitiv}
- b- valoare naturală strict pozitivă deci se disting două clase de echivalența:

  B_1={ b | b număr natural strict pozitiv}

  B_2={ b | b nu este număr natural strict pozitiv}
- c-valoare naturală strict pozitivă deci se disting două clase de echivalența:

  C_1={ c | c număr natural strict pozitiv}

  C_2={ c| c nu este număr natural strict pozitiv}

2.Domeniul de ieşire: Constă într-un număr natural result ∈ [0,c) și este rezultatul procesării numerelor a, b, c. Acesta este folosit pentru a împarți domeniul de intrare în mai multe clase, in funcție de valorile a, b,c:

X_1={ result | GCD(a, b)>c }

X_2={ result | GCD(a, b)==c }

X_3={ result | GCD(a, b) < c }

Clasele de echivalențe globale pe care le avem:

G_111={ (a,b,c) | a ∈ A_1 & b ∈ B_1 & c ∈ C_1 }

G_211={ (a,b,c) | a ∈ A_2 & b ∈ B_1 & c ∈ C_1 }

G_121={ (a,b,c) | a ∈ A_1 & b ∈ B_2 & c ∈ C_1 }

G_112={ (a,b,c) | a ∈ A_1 & b ∈ B_1 & c ∈ C_2 }

G_122={(a,b,c) | a ∈ A_1 & b ∈ B_2 & c ∈ C_2 }

G_212={(a,b,c) | a ∈ A_2 & b ∈ B_1 & c ∈ C_2 }

G_221={(a,b,c) | a ∈ A_2 & b ∈ B_2 & c ∈ C_1 }

G_222={(a,b,c) | a ∈ A_2 & b ∈ B_2 & c ∈ C_2 }

Setul de date se alcătuiește prin alegerea unei valori de intrare pentru fiecare clasa. Exemplu:

g_111: (7,21,5)

g_211: (0,4,2)

g_121: (5,-2,3)

g_112: (3,2,-1)

g_122: (30,5,2)

a	b	c	Rezultat afișat (expected)
7	21	5	2
0	4	2	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”
5	-2	3	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”
3	2	-1	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”
30	5	2	1

```
"""1) Partitionare de echivalenta"""

def test_G111_valid_imputs(self):
    result=ModuloGCD(10, 5, 2).cmmdc()
    self.assertEqual(result,1)

def test_G211_invalid_a(self):
    with self.assertRaises(ValueError):
        ModuloGCD(-5,5,2).cmmdc()

def test_G121_invalid_b(self):
    with self.assertRaises(ValueError):
        ModuloGCD(50,"5",4).cmmdc()

def test_G112_invalid_c(self):
    with self.assertRaises(ValueError):
        ModuloGCD(43,5,-2).cmmdc()

def test_G122_invalid_b_c(self):
    with self.assertRaises(ValueError):
        ModuloGCD(-5,5,2).cmmdc()

def test_G212_invalid_a_c(self):
    with self.assertRaises(ValueError):
        ModuloGCD(100.2,31,None).cmmdc()

def test_G221_invalid_a_b(self):
    with self.assertRaises(ValueError):
        ModuloGCD(-5,32.2,6).cmmdc()

def test_G222_invalid_a_b_c(self):
    with self.assertRaises(ValueError):
        ModuloGCD(-5,15.2,"12").cmmdc()
```

"""1) Partitionare de echivalenta"""

def test_G111_valid_imputs(self):
  result=ModuloGCD(10, 5, 2).cmmdc()
  self.assertEqual(result,1)

def test_G211_invalid_a(self):
  with self.assertRaises(ValueError):
    ModuloGCD(-5,5,2).cmmdc()

def test_G121_invalid_b(self):
  with self.assertRaises(ValueError):
    ModuloGCD(50,"5",4).cmmdc()

def test_G112_invalid_c(self):
  with self.assertRaises(ValueError):
    ModuloGCD(43,5,-2).cmmdc()

def test_G122_invalid_b_c(self):

```
    with self.assertRaises(ValueError):
        ModuloGCD(-5,5,2).cmmdc()

def test_G212_invalid_a_c(self):
    with self.assertRaises(ValueError):
        ModuloGCD(100.2,31,None).cmmdc()

def test_G221_invalid_a_b(self):
    with self.assertRaises(ValueError):
        ModuloGCD(-5,32.2,6).cmmdc()

def test_G222_invalid_a_b_c(self):
    with self.assertRaises(ValueError):
        ModuloGCD(-5,15.2,"12").cmmdc()
```

(b) Analiza valorilor de frontineră (boundary value analysis)

Analiza valorilor de frontieră este adesea utilizată împreună cu partiționarea în clase de echivalență. Această tehnică pune accentul pe testarea valorilor aflate la limita fiecărei clase, deoarece acestea reprezintă frecvent o sursă majoră de erori. În cazul exemplului nostru, odată ce clasele de echivalență au fost stabilite, identificarea valorilor de frontiera devine un proces intuitiv :

- valorile de frontieră sunt 0,1,2 pentru a, b, c

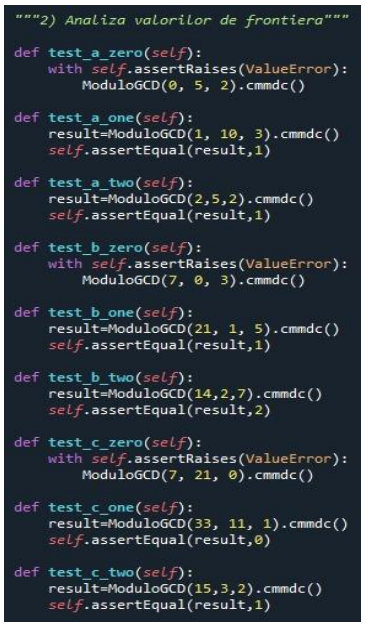
Se vor testa următoarele valori:

- N_1 : 0
- N_2 : 1
- N_3 : 2

Vom avea 9 date de test :

a	b	c	Rezultat afișat (expected)
0	5	2	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”
1	10	3	1

2	5	2	1
7	0	3	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”
21	1	5	1
14	2	7	2
33	11	1	0
15	3	2	1



```
"""2) Analiza valorilor de frontiera"""

def test_a_zero(self):
    with self.assertRaises(ValueError):
        ModuloGCD(0, 5, 2).cmmdc()

def test_a_one(self):
    result=ModuloGCD(1, 10, 3).cmmdc()
    self.assertEqual(result,1)

def test_a_two(self):
    result=ModuloGCD(2,5,2).cmmdc()
    self.assertEqual(result,1)

def test_b_zero(self):
    with self.assertRaises(ValueError):
        ModuloGCD(7, 0, 3).cmmdc()

def test_b_one(self):
    result=ModuloGCD(21, 1, 5).cmmdc()
    self.assertEqual(result,1)

def test_b_two(self):
    result=ModuloGCD(14,2,7).cmmdc()
    self.assertEqual(result,2)

def test_c_zero(self):
    with self.assertRaises(ValueError):
        ModuloGCD(7, 21, 0).cmmdc()

def test_c_one(self):
    result=ModuloGCD(33, 11, 1).cmmdc()
    self.assertEqual(result,0)

def test_c_two(self):
    result=ModuloGCD(15,3,2).cmmdc()
    self.assertEqual(result,1)
```

"""2) Analiza valorilor de frontiera"""
def test_a_zero(self):
  with self.assertRaises(ValueError):
      ModuloGCD(0, 5, 2).cmmdc()

def test_a_one(self):
   result=ModuloGCD(1, 10, 3).cmmdc()
   self.assertEqual(result,1)

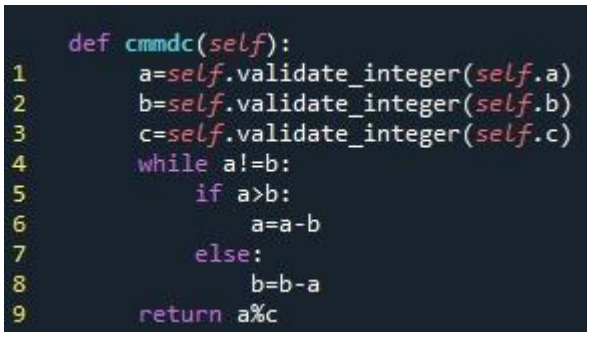
def test_a_two(self):
   result=ModuloGCD(2,5,2).cmmdc()
   self.assertEqual(result,1)

```
def test_b_zero(self):
   with self.assertRaises(ValueError):
      ModuloGCD(7, 0, 3).cmmdc()

def test_b_one(self):
   result=ModuloGCD(21, 1, 5).cmmdc()
   self.assertEqual(result,1)

def test_b_two(self):
   result=ModuloGCD(14,2,7).cmmdc()
   self.assertEqual(result,2)

def test_c_zero(self):
   with self.assertRaises(ValueError):
      ModuloGCD(7, 21, 0).cmmdc()

def test_c_one(self):
   result=ModuloGCD(33, 11, 1).cmmdc()
   self.assertEqual(result,0)

def test_c_two(self):
   result=ModuloGCD(15,3,2).cmmdc()
   self.assertEqual(result,1)
```

2.Testarea structurală

1.Graful de flux de control

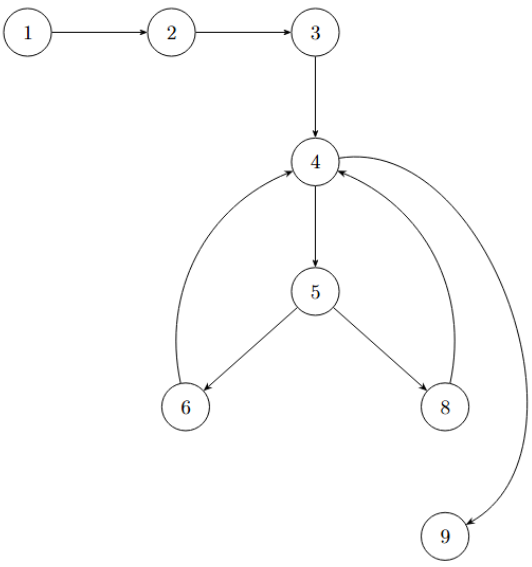
Am ales datele de test astfel încât să parcurgă toate elementele noastre introduse în cod.



```
def cmmdc(self):
    a=self.validate_integer(self.a)
    b=self.validate_integer(self.b)
    c=self.validate_integer(self.c)
    while a!=b:
        if a>b:
            a=a-b
        else:
            b=b-a
    return a%c
```

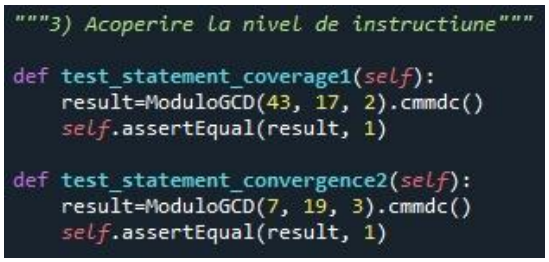
Nodul 1 verifică a, nodul 2 verifică b si nodul 3 verifică c. De asemenea, nodul 4 ne intră pe instrucțiunea while care se duce în if else explicând astfel nodurile 5,6,8. Apoi se întoarce în nodul 4 pana când condiția nu mai este îndeplinita, după care iese cu nodul 9 când condiția din while nu este îndeplinită.

De asemenea,pentru nodurile 1,2,3 este necesara trecerea de catre conditia if not isinstance(value,int) or value < 0 pentru a se putea forma.



(a) Acoperirea la nivel de instrucțiune :

Ca să avem ca rezultat o acoperire la nivel de instrucțiune testul este luat astfel încât să parcurgă fiecare nod al grafului.



```
"""3) Acoperire la nivel de instructiune"""

def test_statement_coverage1(self):
    result=ModuloGCD(43, 17, 2).cmmdc()
    self.assertEqual(result, 1)

def test_statement_convergence2(self):
    result=ModuloGCD(7, 19, 3).cmmdc()
    self.assertEqual(result, 1)
```

"""3) Acoperire la nivel de instructiune"""

```
def test_statement_coverage1(self):
    result=ModuloGCD(43, 17, 2).cmmdc()
    self.assertEqual(result, 1)

def test_statement_convergence2(self):
    result=ModuloGCD(7, 19, 3).cmmdc()
    self.assertEqual(result, 1)
```

a	b	c	Rezultat afișat	Instrucțiuni parcurse
43	17	2	1	1,2,3,4,5,6,4,5,6,4,5,8,4,5,6, 4,5,8, 4,5,8, 4,5,8, 4,5,8, 4,5,8, 4,5,8, 4,5,8,4,9
7	19	3	1	1,2,3,4,5,8,4,5,8,4,5,6,4,5,8,4,5,8,4,5,8,4,5, 6,4,9

(b) Acoperire la nivel de decizie (decision coverage)

	Decizii
(1)	if not isinstance(value,int) or value <0:
(2)	while a!=b:
(3)	if a>b:

```
"""4) Acoperire la nivel de decizie"""

def test_decision_invalid_input(self):
    with self.assertRaises(ValueError):
        ModuloGCD("10",5,2).cmmdc()

def test_decision_loop_skip(self):
    result=ModuloGCD(10,10,12).cmmdc()
    self.assertEqual(result,10)

def test_decisio_loop_true_branch(self):
    result=ModuloGCD(14,10,5).cmmdc()
    self.assertEqual(result,2)
```

"""4) Acoperire la nivel de decizie"""

```
def test_decision_invalid_input(self):
   with self.assertRaises(ValueError):
      ModuloGCD("10",5,2).cmmdc()

def test_decision_loop_skip(self):
   result=ModuloGCD(10,10,12).cmmdc()
   self.assertEqual(result,10)

def test_decisio_loop_true_branch(self):
   result=ModuloGCD(14,10,5).cmmdc()
   self.assertEqual(result,2)
```

a	b	c	Rezultat afișat	Decizii acoperite
“10 ”	5	2	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”	(1)-f
10	10	12	10	(1)-A,(2)-F
14	10	5	2	(1)-A, (2)-A,(3)-A

(c) Acoperire la nivel de condiţie (condition coverage)

Decizii	Condiții individuale
if not isinstance(value,int) or value <=0 :	isintsance(value,int), value <=0
while a!= b:	a!=b
if a>b:	a>b

```
"""5) Acoperire la nivel de conditie"""

def test_condition_isinstance_fail(self):
    with self.assertRaises(ValueError):
        ModuloGCD("9",4,3).cmmdc()

def test_condition_value_fail(self):
    with self.assertRaises(ValueError):
        ModuloGCD(10,-2,3).cmmdc()

def test_condition_true_while(self):
    result=ModuloGCD(4,6,6).cmmdc()
    self.assertEqual(result,2)

def test_condition_false_while(self):
    result=ModuloGCD(3,3,6).cmmdc()
    self.assertEqual(result,3)

def test_condition_true_if(self):
    result=ModuloGCD(4,1,5).cmmdc()
    self.assertEqual(result,1)

def test_condition_false_if(self):
    result=ModuloGCD(1,3,10).cmmdc()
    self.assertEqual(result,1)
```

"""5) Acoperire la nivel de conditie"""

def test_condition_isinstance_fail(self):
 with self.assertRaises(ValueError):
 ModuloGCD("9",4,3).cmmdc()

def test_condition_value_fail(self):
 with self.assertRaises(ValueError):
 ModuloGCD(10,-2,3).cmmdc()

def test_condition_true_while(self):
 result=ModuloGCD(4,6,6).cmmdc()

```
    self.assertEqual(result,2)

def test_condition_false_while(self):
    result=ModuloGCD(3,3,6).cmmdc()
    self.assertEqual(result,3)

def test_condition_true_if(self):
    result=ModuloGCD(4,1,5).cmmdc()
    self.assertEqual(result,1)

def test_condition_false_if(self):
    result=ModuloGCD(1,3,10).cmmdc()
    self.assertEqual(result,1)
```

a	b	c	Rezultat afișat	Condiții/Decizii individuale acoperite
“9”	4	3	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”	isinstance(value,int) -F
10	-2	3	Ridică ValueError: “Valorile trebuie sa fie numere naturale pozitive”	value <=0-F
4	6	6	2	isinstance(value,int) -A, value <=0-A, a!=b-A, a>b-A
3	3	6	3	isinstance(value,int) -A, value <=0-A, a!=b-F
4	1	5	1	isinstance(value,int) -A,

				value <=0-A, a!=b-A, a>b-A
1	3	10	1	isinstance(value,int) -A, value <=0-A, a!=b-A, a>b-F

(d) Testarea circuitelor independente:

Numărul minim de circuite independente pentru a obține o acoperire completă a ramurilor este data de formula lui McCabe. V(G)=e-n+p, unde:

e = numărul de muchii ale graficului.

n = numărul de noduri ale graficului.

p = numărul de componente conectate.

În graful principal din proiectul nostru regăsim 8 noduri iar în cel secundar de verificare există 3.

În cel principal avem 9 muchii iar în cel secundar regăsim 3.

Avem p=4 deoarece flow-ul principal are o componentă normală și 3 ieșiri prin excepții.

Deci, în cazul nostru V(G)= 12-11+4=5

- Circuitele independente:

a) 1/2/3 : isinstance(value,int)=fals -> raises ValueError

b) 1/2/3: value <=0 =fals -> raises ValueError

c) 1/2/3 : isinstance(value,int) și value <=0 =fals -> raises ValueError

d) 1,2,3,4 = fals -> 9

e) 1,2,3,4= adevarat și repetăm 5,6 sau 5,8 până când 4=fals -> 9

```
"""6) Testarea circuitelor independente"""

def test_exit_isinstance(self):
    with self.assertRaises(ValueError):
        ModuloGCD("10",3,6).cmmdc()

def test_exit_value(self):
    with self.assertRaises(ValueError):
        ModuloGCD(10,3.3,6).cmmdc()

def test_exit_both(self):
    with self.assertRaises(ValueError):
        ModuloGCD("10",3.3,6).cmmdc()

def test_exit_a_egal_b(self):
    result=ModuloGCD(10, 10, 12).cmmdc()
    self.assertEqual(result,10)

def test_exit_loop_executed(self):
    result=ModuloGCD(55, 10, 12).cmmdc()
    self.assertEqual(result,5)
```

"""6) Testarea circuitelor independente"""

def test_exit_isinstance(self):
with self.assertRaises(ValueError):
ModuloGCD("10",3,6).cmmdc()

def test_exit_value(self):
with self.assertRaises(ValueError):
ModuloGCD(10,3.3,6).cmmdc()

def test_exit_both(self):
with self.assertRaises(ValueError):
ModuloGCD("10",3.3,6).cmmdc()

def test_exit_a_egal_b(self):
result=ModuloGCD(10, 10, 12).cmmdc()
self.assertEqual(result,10)

def test_exit_loop_executed(self):
result=ModuloGCD(55, 10, 12).cmmdc()
self.assertEqual(result,5)

3.Mutanti

Testarea mutanților este considerată o testare a calității testelor prin adăugarea unor modificări și a unor mutanți în codul de bază.

Mutation= modificare foarte mică

Evaluarea testează dacă se detectează mutanții.

Ideea este de a evalua cât de bune sunt testele făcute.

- Dorim sa testam un mutant care cauzeaza un loop infinit folosind process pentru a rula intr un proces separat motiv pentru care am adaugat functia run in care detectam si omoram rularea fara oprire in siguranta.

```
def run():
    obj=ModuloGCD(10, 10, 5)
    obj.cmmdc_mutant2()
```

def run():

obj=ModuloGCD(10, 10, 5)

obj.cmmdc_mutant2()

- Mutant de ordin 1 - aplica o singura mutație (schimbare), în cazul nostru am schimbat >= înlocuind >

```
"""1) Mutanti de primul ordin"""
def cmmdc_mutant1(self):
    a=self.validate_integer(self.a)
    b=self.validate_integer(self.b)
    c=self.validate_integer(self.c)
    while a!=b:
        if a>=b: # >= in loc de >
            a=a-b
        else:
            b=b-a
    return a%c
```

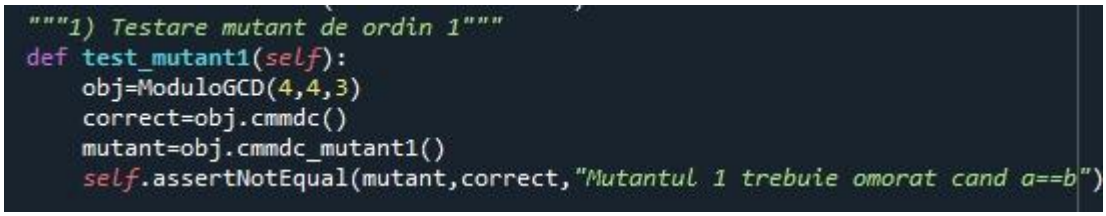
"""1) Mutanti de primul ordin"""

def cmmdc_mutant1(self):

a=self.validate_integer(self.a)

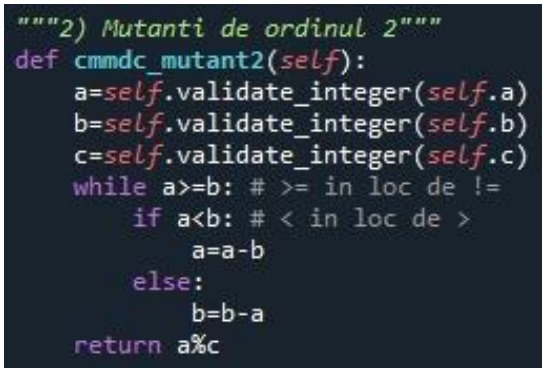
b=self.validate_integer(self.b)

```
    c=self.validate_integer(self.c)
    while a!=b:
        if a>=b: # >= in loc de >
            a=a-b
        else:
            b=b-a
    return a%c
```



"""1) Testare mutant de ordin 1"""
def test_mutant1(self):
obj=ModuloGCD(4,4,2)
correct=obj.cmmdc()
mutant=obj.cmmdc_mutant1()
self.assertNotEqual(mutant,correct,"Mutantul 1 trebuie omorat cand a==b")

- Mutant de ordin 2 - aplica două mutații (simultan) în cazul nostru am schimbat >= înlocuind != și < în loc de >



```
"""2) Mutanti de ordinul 2"""
def cmmdc_mutant2(self):
    a=self.validate_integer(self.a)
    b=self.validate_integer(self.b)
    c=self.validate_integer(self.c)
    while a>=b: # >= in loc de !=
        if a<b: # < in loc de >
            a=a-b
        else:
            b=b-a
    return a%c
```

"""2) Mutanti de ordinul 2"""

def cmmdc_mutant2(self):

a=self.validate_integer(self.a)

b=self.validate_integer(self.b)

c=self.validate_integer(self.c)

while a>=b: # >= in loc de !=

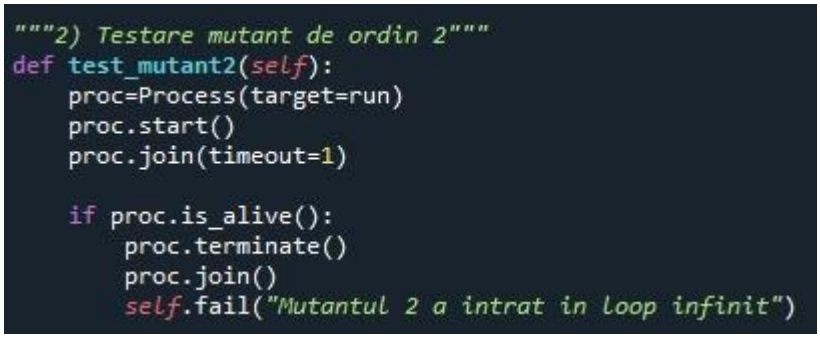
if a<b: # < in loc de >

a=a-b

else:

b=b-a

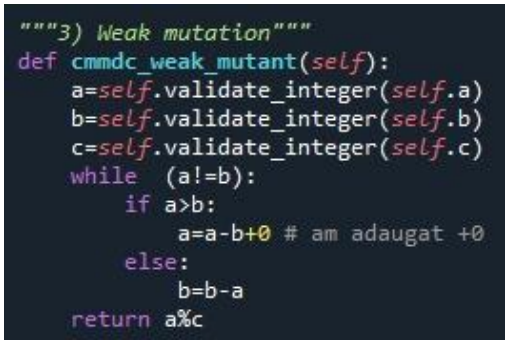
return a%c



```
"""2) Testare mutant de ordin 2"""
def test_mutant2(self):
    proc=Process(target=run)
    proc.start()
    proc.join(timeout=1)

    if proc.is_alive():
        proc.terminate()
        proc.join()
        self.fail("Mutantul 2 a intrat in loop infinit")
```

```
"""2) Testare mutant de ordin 2"""
  def test_mutant2(self):
    proc=Process(target=run)
    proc.start()
    proc.join(timeout=1)

    if proc.is_alive():
      proc.terminate()
      proc.join()
      self.fail("Mutantul 2 a intrat in loop infinit")
```

- Weak mutation - verifica daca starea interna s-a modificat imediat după mutație dar nu neapărat la ieșire, în cazul nostru am adăugat +0, astfel nu modifică ieșirea



```
"""3) Weak mutation"""
def cmmdc_weak_mutant(self):
  a=self.validate_integer(self.a)
  b=self.validate_integer(self.b)
  c=self.validate_integer(self.c)
  while a!=b:
    if a>b:
      a=a-b+0 # am adaugat +0
    else:
```

```
        b=b-a
    return a%c
```

```
"""3) Testare weak mutation"""
def test_weak_mutant(self):
    obj=ModuloGCD(10, 6, 4)
    correct=obj.cmmdc()
    result=obj.cmmdc_weak_mutant()
    self.assertNotEqual(result,correct, "Weak mutation detectata")
```

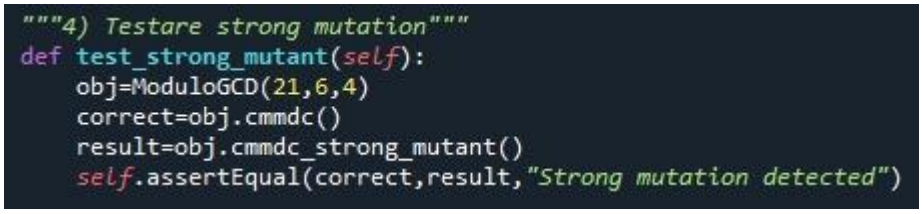
"""3) Testare weak mutation"""
def test_weak_mutant(self):
obj=ModuloGCD(10, 6, 4)
correct=obj.cmmdc()
result=obj.cmmdc_weak_mutant()
self.assertNotEqual(result,correct, "Weak mutation supravietuieste")

- Strong mutation -modificarea afectează rezultatul final al programului, am modificat sa avem a-c în loc de a%c

```
"""4) Strong mutation"""
def cmmdc_strong_mutant(self):
    a=self.validate_integer(self.a)
    b=self.validate_integer(self.b)
    c=self.validate_integer(self.c)
    while a!=b:
        if a>b:
            a=a-b
        else:
            b=b-a
    return a-c
```

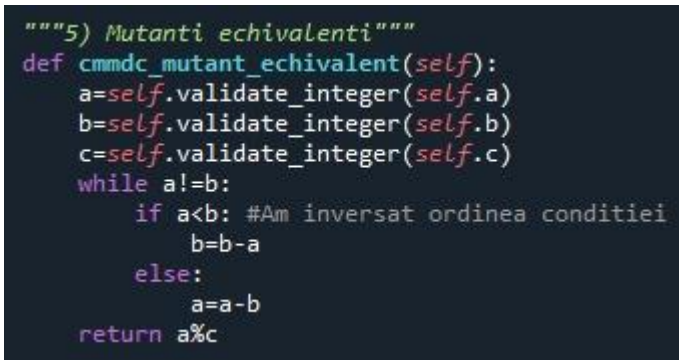
"""4) Strong mutation"""
def cmmdc_strong_mutant(self):
a=self.validate_integer(self.a)
b=self.validate_integer(self.b)
c=self.validate_integer(self.c)
while a!=b:
if a>b:
a=a-b
else:

```
        b=b-a
    return a-c # - in loc de %
```




```
"""4) Testare strong mutation"""
def test_strong_mutant(self):
    obj=ModuloGCD(21,6,4)
    correct=obj.cmmdc()
    result=obj.cmmdc_strong_mutant()
    self.assertNotEqual(correct,result,"Strong mutation detected")
```

- Mutant echivalent - se comportă identic cu programul inițial, deși conține o modificare sintactică, am inversat ordinea condiției




```
"""5) Mutanti echivalenti"""
def cmmdc_mutant_echivalent(self):
    a=self.validate_integer(self.a)
    b=self.validate_integer(self.b)
    c=self.validate_integer(self.c)
    while a!=b:
      if a<b: #Am inversat ordinea conditiei
        b=b-a
      else:
        a=a-b
    return a%c
```

```
"""5) Testarea mutant echivalent"""
def test_equivalent_mutant(self):
    obj=ModuloGCD(32,8,5)
    correct=obj.cmmdc()
    result=obj.cmmdc_mutant_echivalent()
    self.assertNotEqual(correct,result,"Mutantul echivalent ar trebui sa supravietuiasca testarii")
```

"""5) Testarea mutant echivalent"""
def test_equivalent_mutant(self):
obj=ModuloGCD(32,8,5)
correct=obj.cmmdc()
result=obj.cmmdc_mutant_echivalent()
self.assertNotEqual(correct,result,"Mutantul echivalent ar trebui sa supravietuiasca testarii")

Testări suplimentare pentru omorârea a doi mutanţi neechivalenti rămaşi în viaţă:

```
"""6) Testarea mutanti neechivalenti"""
def test_mutant_neechivallent1(self):
    obj=ModuloGCD(12,4,5)
    correct=obj.cmmdc()
    result=obj.cmmdc_mutant_neechivalent1()
    self.assertNotEqual(correct,result,"Mutantul neechivalent 1 trebuie omorat")
```

```
def test_mutant_neechivallent2(self):
    obj=ModuloGCD(14,7,4)
    correct=obj.cmmdc()
    result=obj.cmmdc_mutant_neechivalent2()
    self.assertNotEqual(correct,result,"Mutant neechivalent 2 trebuie omorat")
```

"""6) Testam 2 mutanti neechivalenti"""
def cmmdc_mutant_neechivalent1(self):
a=self.validate_integer(self.a)
b=self.validate_integer(self.b)
c=self.validate_integer(self.c)
while a!=b:
if a>=b:
a=a-b
else:
b=b-a

```
    return a%b #a%b in loc de a%c

def cmmdc_mutant_neechivalent2(self):
    a=self.validate_integer(self.a)
    b=self.validate_integer(self.b)
    c=self.validate_integer(self.c)
    while a!=b:
        if a>=b:
            a=a-b
        else:
            b=b-a
    return a&c


"""6) Testarea mutanti neechivalenti"""

def test_mutant_neechivallent1(self): obj=ModuloGCD(12,4,5)
correct=obj.cmmdc() result=obj.cmmdc_mutant_neechivalent1()
self.assertNotEqual(correct,result,"Mutantul neechivalent 1 trebuie omorat")

def test_mutant_neechivallent2(self):
    obj=ModuloGCD(14,7,4)
    correct=obj.cmmdc()
    result=obj.cmmdc_mutant_neechivalent2()
    self.assertNotEqual(correct,result,"Mutant neechivalent 2 trebuie omorat")
```