HALMSTAD UNIVERSITY

Programming

Lab Sessions

HALMSTAD UNIVERSITY

Assignment V - Presentations

- Zoom room for presentations:

  Meeting ID: 641 7002 9853

  Passcode: 034035

- Order of presentation avaliable.
- **Be on time**
- Martin will be available to solve doubts at the usual meeting room with an exercise list.

HALMSTAD UNIVERSITY

Resources

- Slides for this lecture are available at marcomarinho.com.

Previous Exams

- We will solve questions from previous exams today. They have some math on them =(.
- **I DID NOT MAKE THESE QUESTIONS NOR WILL I MAKE QUESTIONS FOR YOUR EXAM.**
- Focus on how to ignore what's useless and not fall for traps.

200810

- Write a new program that does the same thing as the below but don't use while loops.
  - The program should generate the same output as the below program if the input is the same.

```
tal = int(input("Ange tal"))
x,y,z = 1,1,1
while x <= tal:
    while y <= tal:
        while z <= tal:
            if x <= y and y <= z:
                if x**2 + y**2 == z**2:
                    print("(",x,",",y,",",z,")")
            z+= 1
        y+= 1
        z = 1
    x += 1
    y = 1
```

200810

- Solution: Replace while loops with for loops

```
tal = int(input("Ange tal"))
for x in range(1, tal+1):
    for y in range(1, tal+1):
        for z in range(1, tal+1):
            if x <= y and y <= z:
                if x**2 + y**2 == z**2:
                    print("(",x,",",y,",",z,")")
```

200810

- 4a. what do the two functions foo and bar do?
- 4b. what happens if we remove "-1" on the line ending with -1 and a # in the for loop of bar function
- 4c what is printed if the functions are run?

```
def foo(arg1, arg2):
    f = open(arg1, "w")
    for i in range(arg2):
        for j in range(i):
            f.write("1")
        f.write("\n")
    f.close()


def bar(arg1, arg2):
    foo(arg1, arg2)
    count = 0
    f = open(arg1, "r")
    for line in f:
        count += len(line) -1 #
    f.close()
    return count

print(bar("test", 4))
```

200810

- 4a. foo writes to the file whose name is arg1, it writes lines with an increasing number of characters ‘1’. Starting at 0 characters and ending with a line with arg2-1 characters. bar calls foo to write the file and after opens the file to count the total number of characters that are present of the file, not counting line breaks.
- 4b. The line breaks will also be counted as characters.
- 4c. It prints the total number of characters written to the chosen file. In this case it will print ‘6’.

200810

- 3b#1 What is x = foo(var) if var > 10?
- 3b#2 for what numbers (positive, no decimals), var, must foo(var) return a value and not get stuck in the eternal loop?
- 3b#3 What happens if var is a negative number when calling foo(var)?
- 3b#4 replace the lines of "#" with code so that foo returns a value in the cases where the current implementation of foo does not return a value

```
def bar(value):
    return value*(value-1)*(value-2)

def foo (value):
    #
    #
    #
    while true:
        if value > 10:
            break
        else:
            value += bar(value)
        return(value)
```

200810

- 3b#1 Since nothing will be returned in this case, x will be equal to None.
- 3b#2 The function will never get stuck in an eternal loop.
- 3b#3 The function will run normally.
- 3b#4:

```
if value > 10:
    return value
```

200602

- Consider the function foo_bar below where some parts have been hidden. Specify what should be written behind the hidden parts A, B, C and D, so that the function returns False at the following calls: foo_bar ("1235321"), foo_bar ("anna"), foo_bar ("lol"), foo_bar ("gg"), but returns True at the following call: foo_bar ("1235322"), foo_bar ("else"), foo_bar ("loled"), foo_bar ("stonks").

```
def foo_bar(string):
    string2 = A
    for i in range(len( B )):
        string2 += C
    return D
```

200602

```
def foo_bar(string):
    string2 = ''
    for i in range(len(string)):
        string2 += string[-(i+1)]
    return not string == string2
```

200602

- Consider the example below where a class is found that represents balls (it is optional if you want these to be two-dimensional or three-dimensional). Some parts have been hidden. Replace them hidden parts A, B and C with code so that when the program is run, the volume is printed for one ball with radius 1.

```
import math

class Ball:
    def __init__(self, radius):
        A
    def compute_volume(self):
        return B

    def __str__(self):
        return "Bollen har volymen: " C

boll = Ball(1)
print(boll)
```

200602

```
import math

class Ball:

    def __init__(self, radius):
        self.radius = radius

    def compute_volume(self):
        return (4/3)*math.pi*(self.radius**3)

    def __str__(self):
        return 'The ball has volume: ' + str(self.compute_volume())


boll = Ball(1)
print(boll)
```

200415

I denna uppgift ska du simulera en spindel som klättrar upp på en stege. Så fort spindeln kommer till ett nytt steg på stegen kan den ramla ner till marken igen med en sannolikhet p mellan 0 och 1, eller fortsätta uppåt till nästa steg med sannolikheten 1-p. Stegen innehåller N steg. Om spindeln når det översta steget på stegen så ramlar den ner automatiskt.

a) **(3p)** Skriv en funktion `spindel(p, N)` som returnerar ett trappsteg (representerat som ett heltal) som en simulerad (slumpmässig) spindel ramlar ner ifrån, där `p` och `N` är förklarade ovan. Marken är steg 0, sen följer steg 1, steg 2, …, steg `N`.

b) **(5p)** Skriv en funktion `simulation(p, N, antal_spindlar)` som simulerar antal_spindlar många spindlar som klättrar uppför stegar (och ramlar ner). Funktionen använder sig av funktionen `spindel` och returnerar följande tre saker:
   a. det högsta steg som någon spindel kom till,
   b. medelvärdet av de steg från vilka spindlarna ramlade ner.
   c. Det totala antalet steg som spindlarna klättrade tillsammans.

c) **(12p)** Nu ska du skriva en klass `Spider_climb` som används för mer avancerade simuleringar än de ovan. För klassen gäller följande:
   a. Man ska kunna lägga till och ta bort spindlar.
   b. Varje spindel har en ett eget antal steg på sin stege, dvs. ett eget N som kan skilja sig från andra spindlars N.
   c. För varje spindel är det olika sannolikheter att ramla ner vid de olika stegen. Dessa sannolikheter kan skilja sig mellan spindlarna.
   d. Varje spindel tillbringar en viss tid på varje trappsteg innan den ramlar ner eller fortsätter. Dessa tider skiljer sig mellan steg och spindlar.
   e. Det ska finnas en metod som returnerar spindlarna.
   f. Det ska finnas en metod som simulerar att spindlarna klättrar upp och ramlar ner för stegarna. För varje spindel ska anges hur många gånger den klättrar upp (och ramlar ner). Resultaten ska sparas som instansvariabler (se g. nedan vilka dessa kan vara).
   g. Det ska finnas en metod som visar resultaten av den senaste simuleringen. Förutom a. b. och c. i uppgift b) ovan, ska även det steg som någon spindel spenderade mest tid på visas, samt den totala tiden som spindlarna spenderade på stegarna.
   h. Det ska finnas en metod som tar bort resultaten från den senaste simuleringen.
   i. Det ska finnas en metod som omvandlar objekt av klassen till en sträng där relevant data om objektet återfinns.

```
import random

def spindel(p, N):
    for step in range(N+1):
        fall_down = random.random()
        if fall_down < p:
            return step
    return N

def simulation(p, N, spiders):
    N_max = 0
    N_total = 0

    for spider in range(spiders):
        current_N = spindel(p, N)
        if current_N > N_max:
            N_max = current_N
        N_total = N_total + current_N

    N_average = N_total/spiders

    return N_max, N_total, N_average
```

```
class spider:

    def __init__(self, p, N):
        self.p = p
        self.N = N
        self.time = max_time

    def runSimulation(self):
        results = spindel(p, N)
        return results

class complex_simulation:

    def __init__(self):
        self.spiders = []
        self.lastResults = []

    def addSpider(self, spider):
        self.spiders.append(spider)

    def removeSpider(self, index):
        self.spiders.pop(index)

    def getSpiders(self):
        return self.spiders

    def runSimulations(self):
        temp_result = []
        for spider in self.spiders:
            temp_N = spider.runSimulation()
            tem_result.append(temp_N)
        lastResults = temp_result
        return temp_result

    def getLastSimulation(self):
        return self.lastResults

    def removeLastSimulation(self):
        self.lastResults = []
```

End