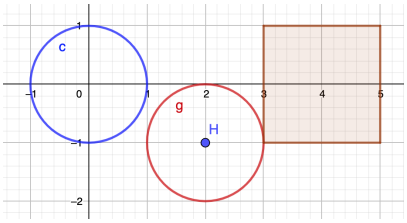
Labb 3 - Geometri OOP

Syftet med den här laborationen är att använda objektorienterad programmering i Python för att designa program med god struktur.



I den här laborationen ska du börja med att planera hur du vill strukturera dina klasser med hjälp av UML och därefter implementera din planering i Python.

Uppgift

Du ska skapa följande geometriska klasser:

- rektangel
- cirkel

Notera att du även kan behöva ytterligare klass(er) om du vill utnyttja arv och/eller komposition.

Klasserna ska ha:

- en area property
- en omkrets property
- en operator overload av `==` för att checka likhet
- en operator overload av komparatoroperatorer `<,>,<=,>=` för jämförelser
- en override av `__repr__()`
- en override av `__str__()`
- x och y som representerar mittpositionen av objektet
- en translationsmetod som gör det möjligt att förflytta x och y
- en metod som checkar om en viss punkt befinner sig innanför i objektet
- felhantering
- en metod som checkar om cirkelinstansen är en enhetscirkel
- en metod som checkar om rektangelinstansen är en kvadrat

Ge klasserna fler lämpliga funktionaliteter som passar, exempelvis skulle det vara trevligt att rita ut geometriska figurerna och se translationer. Dina klasser ska finnas i en .py-fil. Man ska exempelvis kunna använda dina klasser på följande sätt:

```
from geometry_shapes import Circle

cirkel1 = Circle(x=0,y=0, radius=1) # enhetscirkel
cirkel2 = Circle(x=1,y=1, radius=1)
rektangel = Rectangle(x=0,y=0,side1=1, side2=1)
```

```
print(cirkel1==cirkel2) # True

print(cirkel2==rektangel) # False

print(cirkel1.is_inside(0.5, 0.5)) # True

cirkel1.translate(5,5)

print(cirkel1.is_inside(0.5, 0.5)) # False

cirkel1.translate("TRE",5) # ge ValueError med lämplig kommentar
```

Börja med att skapa en planering med diagrams.net och därefter implementerar du din planering i Python. Det är helt okej att använda annan programvara för att skapa UML eller att göra en med papper och penna.

Gör manuella tester, likt ovan, men testa mer. Tips är att ha en .ipynb-fil vid sidan om som importerar dina klasser för att göra manuella tester medan du skriver klasserna.

Bonusuppgifter (frivilliga)

- skapa en kubklass med motsvarande funktionaliteter
- skapa en sfärklass med motsvarande funktionaliteter
- fundera över hur olika metoder, properties förändras ex. area, omkrets
-
- enhetstesta dina klasser
    - ha en separat .py-fil för enhetstesterna

Bedömning

Om du har fått någon kodsnutt från någon annan eller hittat i någon sida är det **viktigt** att du källhänvisar. Skriv en kommentar bredvid koden som du har tagit.

Godkänt

- gjort en enkel planering med UML
- löst uppgift på korrekt sätt
- koden är kommenterad med relevanta kommentarer, docstrings ska användas
- variabelnamnen är bra valda
- gjort flera relevanta git commits

Väl Godkänt

- koden är enkel att följa
- koden är välstrukturerad med lämpliga klasser, funktioner, metoder
- koden återanvänds där det går istället för att återupprepa
- kommentarerna är datavetenskapligt korrekta
- gjort samtliga uppgifter