Programmering og Modellering (PoM)
Ugeseddel 11 — Uge 48 — Deadline Ingen aflevering

Kim Steenstrup Pedersen, Katrine Hommelhoff Jensen, Knud Henriksen,
Mossa Merhi og Hans Jacob T. Stephensen

20. november 2014

1 Plan for ugen

Denne uge tager vi fat på emnet: Objektorienteret design. Objektorienteret programdesign går i sin enkelthed ud på at identificere objekter, som er logiske grupperinger af datarepræsentation og funktionalitet, i det givne problem og benytte disse til at opbygge et modulært opbygget program. Der er forskellige måder at lave objektorienteret programdesign som vil blive berørt til forelæsningerne, herunder anvendelse af såkaldte objektorienteret designmønstre. Et designmønster kan tænkes på som en opskrift på at løse ofte forekommende designproblemer. Derudover skal vi introducere konceptet arv og hvorledes dette anvendes i Python. Arv anvendes til at udvide funktionaliteten og datarepræsentationen af en klasse og vi skal til forelæsningerne se på forskellige eksempler. Nøgleord for denne uge vil være: Arv, objektorienteret design og designmønstre.

Til tirsdag:

Læs Guttag kapitel 8.2, 8.4 og 11.

Til torsdag:

Læs Guttag kapitel 8.2, 8.4 og 11.

Derudover kan man læse uddrag fra http://sourcemaking.com /design_patterns efter behov og du finder konkrete Python eksampler på design patterns på https://github.com/faif/python-patterns.

Orienterer dig også i eksamenopgaveformuleringen fra PoP eksamen 2013-2014, som du finder under Undervisningsmateriale/Eksempel på eksamensopgave.

Bemærk: Der er ingen ny obligatorisk afleveringsopgave i denne uge da forrige uges opgave fortsætter ind i denne uge og afleveres fredag d. 28/11 2014.

1.1 Tirsdagsøvelser

Besvarelser af disse opgaver skal ikke afleveres, men opgaverne forventes løst inden torsdag.

11ti1 Skriv en klasse som skal repræsentere et datoobjekt som indeholder attributter `day`, `month` og `year`. Skriv en metode `increment_day` til klassen som tager et heltal `n` som parameter og fungere således at objektet vi kalder metoden på skal have sin dato fremskrevet med `n` dage. Husk på at der er forskellig antal dage i månederne. Udfordring: Håmdtere din funktion skudår korrekt? (Se `http://wikipedia.org/wiki/Leap_year`)

11ti2 Skriv en afprøvning af metoden `increment_day` fra opgave 11ti1. Afprøvningen skal designes således, at alle skrevne programlinjer bliver afprøvet (også kaldet en *intern afprøvning*), samt at specifikationens grænsetilfælde afprøves (også kaldet en *ekstern afprøvning*).

11ti3 Skriv en klasse `KompleksPoly` som skal repræsentere et kompleks polynomium. Lav konstruktørmetoden `__init__` således at man kan angive polynomiets koefficienter som parametre. Vælg en passende polynomium orden og koefficienter i det tilfælde at der ikke angives en parameter. Du kan vælge at repræsentere polynomiets koefficienter med Pythons indbyggede datatype `complex` eller du kan anvende klassen `Kompleks` vi udviklede under forelæsningen i sidste uge. Tilføj en `__str__` metode som returnere en pæn strengrepræsentation af polynomiet som du selv vil skrive det op på papir. Tilføj en metode `evaluate` som tager en parameter `z` og evaluerer polynomiet i den kompleksværdi angivet ved `z`.

11ti4 Skriv en afprøvning af klassen `KompleksPoly` fra opgave 11ti3. Afprøvningen skal designes således, at alle skrevne programlinjer bliver afprøvet (også kaldet en *intern afprøvning*), samt at specifikationens grænsetilfælde afprøves (også kaldet en *ekstern afprøvning*).

11ti5 Lad os forestille os at vi skal skrive et program som kan repræsentere en skål med farvede glaskugler og som kan sortere kuglerne over i nye skåle, således at hver skål repræsentere en unik farve. Prøv på papir at identificere mulige objekter / klasser som du vil kunne anvende til at løse dette problem. For hvert objekt du identificere prøv at opskrive de attributter og metoder du kan forestille dig at objektet skal have (du behøver ikke at implementere metoderne).

11ti6 Lad os vende tilbage til den obligatoriske opgave på ugeseddel 7 — Integration ved Riemannsummer — hvor vi implementerede to forskellige metoder. Prøv at designe en klasse til at repræsentere reelle funktioner af én variabel og tilføj passende metoder til klassen. Tilpas derpå din løsning til funktionerne `rInt` og `rIntMid` fra ugeseddel 7, således at de bliver til metoder på din nye funktionsklasse og anvender metoder og attributter på klassen.

1.2 Torsdagsøvelser

Besvarelser af disse opgaver skal ikke afleveres, men opgaverne forventes løst inden tirsdag i efterfølgende uge.

11to1 Lad os vende tilbage til funktionsklassen fra opgave 11ti6 (hvis du ikke allerede har løst opgave 11ti6 så gør det nu inden du løser denne opgave) og lad os sige at vi ønsker at udvide funktionaliteten af klassen med en implementation af Monte Carlo integrationsmetoden fra opgave 7to4. Vi kunne gøre dette ved at tilføje en ny metode til vores funktionsklasse, eller vi kan benytte os af nedarvning af funktionalitet og lave en ny klasse som arver fra vores funktionsklasse og derudover inkludere en metode `mcInt`. Prøv at implementere denne nye klasse. (Dette er i virkeligheden ikke en særlig god programdesignstrategi så vi vender tilbage til dette problem i opgave 11to4)

11to2 Forestil dig, at du skal lave et objektorienteret programdesign som skal kunne repræsentere datasæt af forskellig karakter, således at antallet af målepunkter og antallet af målte værdier pr. målepunkt kan variere. Forestil dig ligeledes at du kan hente data ind på forskelligvis — eksempelvis ved 1) indtastning på tastaturet, 2) indlæsning fra fil, 3) hent fil fra internettet, 4) hent fra en database. Som eksamenpler på datasæt kan du kigge på flueaeg.txt og pattedyr.txt du finder i Undervisningsmateriale/ Datamateriale. Prøv at udarbejd et objektorienteret programdesign som repræsentere abstraktionen datasæt som beskrevet ovenfor. Hint: Du kan med fordel benytte nedarvning af funktionalitet.

11to3 Implementér dit programdesign fra opgave 11to2, således at du eksempelvis kan indlæse de to datasæt flueaeg.txt og pattedyr.txt eller kan indtaste et datasæt fra tastaturet (hint: Her kunne du lade hver linje repræsentere et nyt målepunkt og lade de målte værdier være separeret af mellemrum).

11to4 De to designstrategier vi anvendte til at udvikle funktionsklassen i opgave 11ti6 og 11to1 er fine nok for små programmer og i situationer, hvor man ikke forventer mange flere udvidelse med nye metoder. Er man derimod i en situation hvor man kan forestille sig mange nye metoder, er disse to designstrategier ikke særlige hensigtsmæssige (vi kan eksempelvis ende ud med

meget lange nedarvningshierarkier som kan være svære at overskue). En alternativ designstrategi er at anvende det såkaldte strategimønster (Eng. Strategy pattern) som omtaltes ved forelæsningerne (og se evt. også `http://sourcemaking.com/design_patterns/strategy`). I denne sammenhæng er en strategi en konkret integrationsmetode. Lad os først definere en grænseflade (Eng. interface) som vores strategier skal implementere:

```
class IntegrationStrategyInterface(object):
   """This class represents the interface for integration strategies
         following the Strategy design pattern."""
   def integrate(self, f, a, b, n):
       """f is a reference to a function object which has an evaluate metode,
           a and b are start and end values for the integration interval,
           n is number of partitions / samples.
           The method should return the result of the integrationmetode"""
       pass
```

Bemærk at jeg medvilje har valgt at lave metoden `integrate` tom ved at anvende `pass` — jeg ved endnu ikke hvilken integrationsmetode det drejer sig om. Du kan nu implementere integrationsmetoderne ved, for hver metode, at lave en integrationsstrategiklasse som arver fra `IntegrationStrategyInterface` og inkluderer selve implementationen af metoden i `integrate`, eksempelvis således:

```
class RiemannSumIntegration(IntegrationStrategyInterface):
   def integrate(self, f, a, b, n):
       # Do the actual computations here and return result
       return result
```

Omskriv din funktionsklasse således at du fjerner alle integrationsmetoderne og i stedet kan angive en konkret integrationsmetode ved at instantiere den tilhørende integrationsstrategi klasse. Måske har du behov for en metode på funktionsklassen, som sætter den aktuelle integrationsmetode. Du skal implementere et par integrationsmetoder som strategier. Hint: Hvis du har mistet overblikket, så prøv at begynde med at tegne et klassediagram som illustrere alle klasser i problemet og deres relationer (eks. arv og has-an-instance-of) — du kan anvende fattigmands UML notationen der blev introduceret til forelæsningen.

11to5 Til forelæsningerne gennemgik jeg en implementation af en simulator af solsystemet som anvendte Model-View-Controller designet. Download løsningen fra Absalon — du finder den under Undervisningsmateriale/Eksempel på eksamensopgave/. Kig de forskellige moduler igennem og dan dig et overblik over hvorledes løsningen er bygget op. Du kan med fordel kigge på klassediagrammet imens du læser koden igennem.

11to6 Tag udgangspunkt i din implementation af et datasæt fra opgave 11to2 og lad os tænke på dette som vores model. Prøv at følge Model-View programdesignstrategien og omskriv din datasæt repræsentation så den kan fungere som en model. Tilføj nu konceptet View til dit programdesign således at du kan lave forskellige visualiseringer af datasættet. Prøv at implementere nogle konkrete Views som eksempelvis et View der sætter datasættet pænt op og udskriver det i terminalen, og et View som laver en graf af datasættet (hint: Hvis du anvender flueaeg.txt og pattedyr.txt, som begge indeholder to målte værdier pr. målepunkt, så kan du blot vise disse som punkter i et 2-dimensionelt koordinatsystem).

11to7 For de avancerede: Du kan få din klasse `KompleksPoly` fra opgave 11ti3 til at fungere som en funktion ved at tilføje den specielle metode `__call__(self, z)` og lade denne metode kalde metoden `evaluate(self, z)`. Du kan nu skrive kode som:

```
f = KompleksPoly()
z = complex(1.0, 2.0)
q = f(z)
```

Prøv at implementere dette på `KompleksPoly` klassen.