Velkommen til forkurs i python for MENA og LAP

Opplegget

- Mandag til torsdag: To timer forelesning - fire timer programmering
- Torsdag og Fredag: En større oppgave om differensiallikninger
- Leselekser: Les kapittelet før hver forelesning
- Skrivelekser: Noen oppgaver til å programmere for hånd
- Ikke alt vil bli sagt på forelesning, noe må du finne fram til i tabeller selv, det forventes at du leser

Dag 1: Det grunnleggende

Hva er programmering?

- Programmering - å snakke med datamaskinen, programmeringsspråk - et språk datamaskinen forstår. Vi lærer språket python
- Forskjellen på menneskespråk og dataspråk: Datamaskinen er en pedant
- Å programmere er et håndverk, du må gjøre det
- Alle gjør feil, en god programmerer er en som ikke gir opp!

Å komme i gang

- Eksempel utregning av høyden til en ball:

$$y(t) = v_0 t - \frac{1}{2} g t^2 \text{gir :} \tag{1}$$

$$y(0.6) = 5 * 0.6 - \frac{1}{2} * 9.81 * 0.6^2 \tag{2}$$

$$\tag{3}$$

- Vi kan få datamaskinen til å regne ut dette
- To måter å kjøre python på

Gjenbruk av størrelser - variable

Variable - ord

- Vi oppretter variable for å gjenbruke størrelse. De lagres i minnet.
- Programmet blir lettere å lese, ikke for datamaskinen, men for deg og meg - lag fornuftige navn
- Mange variabeltyper for ulike typer ting
- I python forstår datamaskinen variabeltypen utfra skrivemåten når vi setter verdien første gang. Navnet settes lik en verdi på høyresiden av et likhetstegn

Variable

Int - heltall, float - desimaltall

- int, integer, for heltallsverdier
- Vi initialiserer en int ved:
- float, flyttall, for desimaltall.
- Må deklareres med komma:
- Finnes også type for komplekse tall (complex)

bool - sannhetsverdier

- bool kan ha verdiene True (Sant) og False (Falskt)
- Vi kan initialisere:
- Vanligst ved uttrykk. Vi kan regne ut en sannhetsverdi

Tegn	Betydning	Tegn	Betydning
==	Er lik	!=	Er forskjellig fra
<	Mindre enn	<=	Mindre eller lik
>	Større enn	>=	Større eller lik

Variable

str - string - tekststrenger

- str Tekstverdi
- Vi kan initialisere: Husk anførselstegn eller ’
- Uforanderlig (den gamle verdien blir i minnet)

Variable

Tupler - rekker av verdier

- En rekke verdier av ulike typer i parentes og separert med komma
- Vi får fatt i hvert enkelt element og biter ved indeksering.
- Indeksering starter på 0. Baklengs fra -1. Lengde fra len().
- Sendes rundt inn i programflyten. I tekststrenger til formatering

Variable

Lister - list

- Den foranderlige broren til tuppelet
- I klammeparentes. Ellers svært lik
- Kan itereres over (skal vi snakke om senere)
- Nestede lister, lister av lister, dobbelt indeksering
- Vi kan legge inn ny elementer tilslutt med append(element). Vi kan slette elementer med del liste[indeks]
- Liste med like elementer, [0]*10. Liste med etterfølgende heltall range(m,n,k)

Flere variable dict og none

- Dict likner list, men vi bestemmer også indeksen, kan være et hvilketsomhelst objekt
- Slettes på samme måte, legges til ved å sette ordbok[ny_indeks] = nytt_element
- eng. None type ingen type. Oppstår når verdien ikke er satt.
- Vi går ikke nærmere innpå disse typene. Du kan selv lese om dem

Variable

Konvertering mellom variable - casting

- Skriv: castet_variabel = ny_type(variabel)
- Særlig aktuelt mellom heltall og flyttall. For å unngå heltallsdivisjon
- Heltallsdivisjon: Resten kastes 3/2 = 1
- Heltallsdivisjon kan også unngås ved å sette tallet til flyttall i utgangspunktet 3./2 = 1.5. Vær forsiktig med heltallsdivisjoner!
- Casting også vanlig til og fra tekststrenger. (Vi ser på dette senere i forbindelse med brukerinput)

If- forgreininger og løkker

If-forgreininger

- For å kunne behandle ulike utfall ulikt
- Eksempel ball som faller. Vi vil bare skrive ut høyden dersom den er over bakken.
- Kan også behandle resten av tilfellene med else (ellers). Dersom vi ønsker en egen melding om ballen allerede har landet
- Vi kan behandle flere enn to tilfeller med elif. Tenk om vi vil ha spesialbehandling når ballen traff bakken nå.

If- forgreininger og lÃ¸kker

while-løkker

- Vi vil gjøre ting igjen og igjen, kanskje til noe forandrer seg.
- Tenk at vi vil simulere ballens fall, d.v.s. finne høyden hvert sekund, inntil den når bakken
- Dette gjør vi med en while-løkke. Setningene i blokken blir utført igjen og igjen inntil ballens høyde er null.
- Kan også kombineres med tellevariabel hvis vi vil gjøre noe et visst antall ganger, iterere over elementene i en liste

.

If- forgreininger og løkker

for-løkker

- Spesialisert løkke for å iterere over liste/tuppel eller gjøre noe et bestemt antall ganger
- Vi viser eksempel som lager pen utskrift av simuleringsdataene vi fant over:
- for-løkka gir kortere kode, det er også lettere for mennesker å se hva som foregår

Dag2: Funksjoner og pakker, brukerinput og feilmeldinger

Funksjoner

- Funksjoner i python likner på funksjoner i matematikken.
- Funksjoner er definisjoner på hva som skal gjøres (funksjonsblokken) med noen parametre vi sender inn og hva som skal komme ut. Som en boks som produserer noe fra noe (annet):
- En funksjon begynner med def, så funksjonsnavn, en parentes med navn på parametervariablen og et kolon. Så kommer selve funksjonsblokken.
- Funksjonen kjøres bare når den kalles
- Vi må definerer funksjonen før vi kan kjøre den. (Høyere opp i programmet.)

Funksjoner

Hva er poenget?

- De fleste funksjoner returnerer noe, det skriver vi etter ordet retur. Verdien som sendes med der byttes ut med funksjonskallet.
- Vi kan også skrive funksjoner som ikke returnere noe. De kan gjøre noe vi skal gjøre igjen og igjen. Regne ut noe, for så å skrive det ut e.l.
- Med funksjoner kan vi sortere ut oppgaver som hører til for seg.
- $\rightarrow$ koden blir ryddigere, vi får bedre oversikt.
- De beste funksjonene er relativt korte og oversiktelige, blir det for langt, del opp

Pakker/moduler

Hvordan og hvorfor importere?

- Ikke alle må finne opp kruttet, pånytt
- Ofte er de ferdiglagde funksjonene raskere enn det vi kan skrive selv. (python er et tregt språk, pakker skrevet i c,c++ eller FORTRAN)
- Vi viser fire måter å importere på; Eksempel: import av math:
- To måter å importere alt på
- To måter å importere noe på

Pakker/moduler

Hvilke pakker finnes, og hvordan finner vi ut mer om dem?

- Det finnes veldig mange funksjoner, i dette kurset skal vi lære om/ bruke:
- Standard python:
- math, random, sys, time
- Tileggspakker:
- numpy, scitools
- Finne ut mer: pythons hjemmeside, google. Lurer du på hvordan, har noen andre sannsynligvis gjort det før.

Feil og feilmeldinger

Hjelp jeg har fått en feilmelding!

- Når du programmerer vil du alltid få feil i koden. Sannsynligvis har du erfart dette allerede
- Pust rolig! Les feilmeldingen!
- I feilmeldingen står:
- Linjenummer, gjentagelse av det gale utsagnet med en liten pil
- Feiltype med en liten utdypende setning
- I emacs står linjenummeret nederst. Gå til riktig linje og les feilmeldingen igjen. Forstår du nå.

Feil og feilmeldinger

Hva har gått feil? Viktige feiltyper

- IndentatioError: Innrykket er feil
- IndexErrror: Du prøver å få ut element utenfor liste
- NameError: Variabelnavn er ikke brukt før, skrevet feil, du mangler import eller anførselstegn.
- SyntaxError: Du mangler kolon før blokk, har glemt å avslutte parentes e.l. Feilmelding kan si det står lengre ned

Feil og feilmeldinger

Hva har gått feil? Viktige feiltyper

- TypeError: Du gjør noe med variabel av feil type. Du har glemt å caste e.l.
- ValueError: Variabelen har ulovelig verdi. log av negativt tall, forsøker å caste ”heitil et tall
- Se Sindres feilmeldings- og debuggings-filmer på http://folk.uio.no/sindrf/python/

Brukerinput - raw_input og sys

Hvorfor spørre brukeren, og hvordan?

- For enkelt å endre programmet, variable som skal endres, interaktivitet, programmer for andre.
- Vi lærer to måter å gjøre det på
- Felles: vi tar alltid inn input som tekststreng
- $\longrightarrow$ Vi må caste variable fra brukeren
- Brukeren vet ikke alt, ting kan gå feil
- $\longrightarrow$ Vi trenger feilhåndtering

Brukerinput - raw_input og sys

raw_input - Å spørre brukeren direkte

- Med raw_input spør vi brukeren inne i programmet. (Der vi trenger variablen.)
- Eksempel, vi spør brukeren om ballens startfart:
- Vi caster variabelen til float
- Vi kan også evaluere uttrykk fra brukeren ved hjelp av eval
- Problemer med eval: Vi mister kontroll over hva vi tar inn. 'Alt er lov'. Vanskeligere å feilhåndtere. Bruk eval bare når du trenger det.

Brukerinput - raw_input og sys

sys - Å oppgi brukerinput før kjøring

- Brukeren må oppgi input før kjøring starter
- Vi får tak i verdiene fra sys.argv
- sys.argv er array med alle ordene etter python, sys.argv[0] er programnavnet
- Mer som kan gå galt, mindre brukervennlig, raskere. Egner seg bedre for programmer som brukes av 'eksperter'.
- Må importere pakken sys (men der er det også annet snadder)

Brukerinput - raw_input og sys

Brukeren er dum, hva gjør brukeren feil?

- Bukeren kan oppgi feil objekttype $\rightarrow$ ValueError.
- Brukeren kan glemme å oppgi verdi (særlig ved kommandolinjeinput) $\rightarrow$ IndexError.
- Vi håndterer feilene ved hjelp av try-except-grener.
- Lurt å teste spesifikt for hver feil, (så vi ikke har syntaksfeil i uttrykket vi tester på)
- Test bare feilen, ikke hele programmet

Brukerinput - raw_input og sys

Brukeren er dum, feilen er funnet, hva nå?

- Hva gjør vi når vi har funnet feilen? Vi kaster ut dumme brukere v.h.a. sys.exit(1)
- Vi går i løkke til brukeren gjør det riktig,
- Vi håndterer valgfri informasjonsoppgivelse
- Det siste er lettere med pakken getopt. Se læreboka til INF1100 hvis du vil lære mer om denne pakken.
- (Siden det meste går greit med eval, kan vi ikke teste feilen med en gang, den kommer senere eller vi får flyttall istedenfor heltall)

Tilbake til lister, list comprehensions

- NumPy er først og fremst en pakke som gir oss arrayer
- Først lærer vi hvordan vi på kortform kan initialisere en liste til en eller annen funksjonsverdi - med en implistitt for-løkke
- Dette er list comprehensions:
- Tenk deg at vi vil lage en tabell med teperaturer i celsius og farenheit

NumPy arrayer

Hva skiller en NumPy-array fra en liste?

- På mange områder er NumPy-arrayer og lister nesten helt like
- Forskjeller: arrayer kan kun ha objekter/elementer av en type. For eksempel kan alle være float, alle int eller alle str
- Vi initialiserer NumPy arrayene på en annen måte, for det første må vi ha NumPy-pakken
- Vi må vite hvor lang NumPy-arrayet skal være når vi initialiserer den. Det koster mye å legge inn elementer etterhvert. (arrayet er ikke uforanderlig, men lengden er det.)
- Å bruke NumPy-arrayer er mer effektivt, raskere språk kjører i bakgrunnen

NumPy arrayer

Hvordan lager vi en NumPy array?

- Vi importerer først numpy ved å skrive:
  from numpy import *
- NumPy-modulen inneholder alle funksjonene og variablene i math. Du trenger derfor ikke importere math på nytt når du importerer NumPy
- Vi får en NumPy array ved å:
- caste en passende liste til en array
- parallellen til range - linspace. linspace tar også flyttallsverdier
- Vi får n nuller i arrayet ved å skrive zeros(n)

NumPy arrayer

Hva kan vi gjøre med arrayer?

- Med arrayer/vektorer følger vektoraritmetikk.
- Det betyr at vi kan anvende en funksjon på alle elementene i arrayet, ved å anvende funksjonen på hele arrayet. (Vi sparer for-løkker)
- P.g.a. de raske språkene i bakgrunnen går dette raskere enn med vanlige for-løkker
- fra NumPy får vi også xrange. Denne likner på range, men kan bare brukes til å itereres over, altså i for-løkker. Til gjengjeld får xrange fart på for-løkka

NumPy arrayer

Mulige problemer, hva må vi være obs på

- Dersom en funksjon er stykkevis bestemt, må vi vektorisere funksjonen før vi anvender den
- Dersom vi caster fra en liste, må listen ha de riktige egenskapene (Dette er som regel ikke noe problem)
- Må vite lengden på arrayet før vi oppretter det. Dersom dette ikke er mulig, er løsningen som oftest å lage en liste der vi legger inn elementer med append, og deretter caster til array.
- Ellers kan vi bruker arrayer til så og si det samme som lister. Vi skal se at vi kan bruke dem til å lage plot, når vi kombinerer dem med plotteverktøyet i scitools

Scitools og plotting

Scitools - en monsterpakke

- Når vi skriver from scitools.all import *, importerer vi all funksjonalitet i NumPy, pluss mye mer.
- Scitools gir oss plotteverktøy som gjør at vi kan skrive plotteprogrammer som bruker plotteprogrammet i systemet vi er i.
- Syntaksen i Scitools er nesten identisk med syntaksen for plotting i Matlab. Når du lærer deg hvordan du bruker scitools, lærer du deg dermed også hvordan du kan plotte i Matlab.

Så hvordan plotte?

- For å plotte en ”vanligfunksjon, må vi ha informasjon om punktene på x-aksen, og de tilsvarende punktene på y-aksen. Til dette bruker vi arrayer
- Punktene må ligge tett nok til at funksjonen ser pen ut, men ikke overdriv.
- Vi viser hvordan vi kan lage et meget enkelt plott:
- Så tar vi kontroll over detaljene

Å kontrollere detaljene i plottet

- Det er svært mange ting ved plottet vi kan bestemme. De viktigste er funksjoner/parametere i parentes:
- Tittel (title)
- tekst på x- og y-akse (xlabel, ylabel)
- Tekst til hver enkelt funksjon (legend)
- Vi kan lagre plottet (hardcopy)
- Farge og form på grafene (se tabell i heftet)
- Akselengder

Scitools og plotting

Flere muligheter

- Vi kan plotte flere funksjoner i samme plott.
- For sammenlikning av numerisk og analytisk løsning.
- Sammenlikning av ulike data
- Plott fra funksjon i modell, og hvordan disse passer til reelle data
- Hvis vi lurer på noe kan du i interaktiv økt:
- Skrive help(plot) eller pydoc scitools.easyviz.plot
- Kjøre tutorialen: pydoc scitools.easyviz

Scitools og plotting - film

Mange plott - en film?

- Dersom vi har en funksjon av to variable, kan vi lage en film av funksjonen
- Dersom ingen av de variable er tid, kan vi velge en som blir tid i filmen, ellers velger vi selvsagt tiden som tid
- Alle filmer er massevis av bilder vist etter hverandre
- Vi lager en film av en funksjon ved å ta slike bilder, lagre dem og sette sammen filmen.
- Hvert plott er et bilde.

Film

Viktige elementer når vi lager en film - å lagre plottene

- Vi må lage en løkke over tid, for hver runde lager vi et plott
- Hvert plott må ha et unikt navn, slik at de ikke overskriver hverandre.
- Samtidig må navnene likne, slik at vi lett kan sortere ut bildene til filmen
- En alfabetisk sortering av plottefilene, må gi den kronologiske rekkefølgen
- $\longrightarrow$ Vi gir hvert plottene navn på formen navn0000.eps, navn0001.eps, o.s.v.
- Dette får vi til ved en tellevariabel, her teller, vi lagrer da med: hardcopy('navn%04d.eps'%teller)

Film

Viktige elementer når vi lager en film

- Vi må passe på at aksene er like i hvert plott, ellers vil filmen se merkelig ut
- Lag selve filmen etter at vi er ferdig med plottene, altså utenfor løkken
- Default-formatet på filmen er som regel avi, endre dette gjerne til gif
- Vi lager selve filmen ved å skrive
- movie('navn*.eps', encoder='convert', fps='4', output_file='filmnavn.gif')
- Det tar litt tid å lage en film, datamaskinen har ikke kræsjet (tips, ikke lag for mange plott som skal inn i filmen din)
- Vi viser filmen ved å skrive: animate filmnavn.gif i terminalvinduet

Tilfeldige tall

Hva er tilfeldige tall på datamaskinen?

- På datamaskinen kan vi få generert tilfeldige tall ved å bruke pakken random.
- Datagenererte tilfeldige tall er aldri virkelig tilfeldige, de er generert av en funksjon.
- Tilsynelatende tilfeldige tall - pseudotilfeldige tall.
- Vi kan ha en uniform distribusjon - alle områder i intervallet er like sannsynlige.
- Andre distribusjoner: Gaussisk distribusjon, kontinuerlige, punktdistribusjoner

Tilfeldige tall

Tilfeldige tall i python - hva kan vi bruke dem til?

- Den vanligste funksjonen for tilfeldige tall, gir tilfeldige tall i en uniform distribusjon i intervallet $[0, 1)$
- Vi importerer random: import random
- Vi får et nytt tilfeldig tall ved å be om random.random
- Vi kan får uniformt distribuerte, tilfeldige tall i intervallet $[A, B]$ ved å kalle på a = random.uniform(A, B)
- Vi kan bruke tilfeldige tall til å simulere sannsynlighetsbestemte hendelsesforløp. F.eks. terningkast.
- Hvis vi bruker tilfeldige tall i $[0, 1)$, kan vi bruke det vi vet om sannsynligheter i intervallet til å lage simuleringen.

Tilfeldige tall

Tilfeldige tall i python - hva kan vi bruke dem til?

- I python kan vi også generere tilfeldige heltall. Det kan gi oss en enklere versjon av programmer der vi simulerer endelige utfall:
- For å få et tilfelldig heltall i intervallet $[A, B]$, bruker vi random.randint(A,B).
- Vi kan også bruke tilfeldige tall til å simulere andre ting en spillliknende situasjoner.
- Bevegelsene til partikler i et statistisk ensemble og liknende, er det lett å se at vi kan simulere på denne måten
- Vi kan også løse problemer som ikke er løselige analytisk. Blant annet kan vi regne vanskelige integraler, og gjøre simuleringer med de såkalte Monte-Carlo-metodene

Tilfeldige tall

Eksempel: Virrevandiring

- Virrevandrere er partikler som beveger seg tilfeldig rundt. Vi kan tenke på dem som gassmolekyler som beveger seg rundt.
- Tenker oss først en endimensjonal virrevandring. Partiklen kan bevege seg til høyre og venstre med lik sannsynlighet.
- Vi kan plotte bevegelsene.
- Legger til flere virrevandrere
- Vi er interessert i gjennomsnitt og standardavvik
- Flere dimensjoner?