

# TDT4120

## Øvingsforelesning 1

# Introduksjon til Python

Basert på foiler av Åsmund Eldhuset

Presentert av Martin Gammelsæter

# Python!

- “A C program is like a fast dance on a newly waxed dance floor by people carrying razors.”
- “C++: Hard to learn, and built to stay that way”
- “Java is, in many ways, C++--”

# Hello World i Java

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

# Hello World i Python

```
print "Hello World!"
```

# Installasjon og verktøy

- De aller fleste Linux-distribusjoner, BSD-varianter og OS X-versjoner har Python installert fra før
- På Windows må du besøke denne siden: <http://python.org/download/> og laste det ned
- Eventuelt så finnes det tilgjengelig på stud via ssh
- For å skrive Python kan du bruke hvilken teksteditor du vil; vim, emacs, TextMate, PyDev(Eclipse), PyCharm osv.

# Grunnleggende om Python

- Språket er tolket (“interpreted”), ikke kompilert
  - Koden oversettes til maskinkode mens den kjøres, noe som fører til at Python er tregt.
  - Syntaksfeil oppdages ikke før man “når” de i koden.
- Dynamisk typing
  - Variabler deklarerer ikke, men opprettes automatisk første gangen de brukes.
  - Typen til en variabel kan endre seg underveis, avhengig av hva slags data du legger i den
- Enkel syntaks

# Generell syntaks

- Kommentarer startes med #
- Semikolon brukes ikke
- Indentering og linjeskift har noe å si
  - Må du dele opp en linje, gjøres det ved å sette en \ før linjeskiftet
- Blokker ({} i Java) startes med et kolon, og indenteringen er det eneste som avgjør hvor blokken slutter
  - All kode på samme nivå må ha samme indentering. *Ikke bland mellomrom og tab!*

# Om interpreteren

- Startes ved å logge inn på stud og skrive `python`
- Her kan du skrive inn og utføre én kodelinje av gangen, eller større blokker
- Fin til å eksperimentere med
- Veldig fin kalkulator!
- Skal du lage et større program, bør du lagre det i en tekstfil, f.eks. `program.py`, og kjøre det slik:  
`python program.py`
- Hvis du vil kjøre programmet mot den samme input'en flere ganger, kan du lagre input'en i en tekstfil og kjøre det slik:  
`python program.py < input.txt` (Python-programmet spiser input-fila, nam nam :)

# Utskrift til skjermen

- `print x` skriver ut verdien av  $x$  etterfulgt av linjeskift
- `print a, b, c` skriver ut  $a$ ,  $b$  og  $c$  med mellomrom imellom på samme linje, etterfulgt av linjeskift
- `print a, b, c,` gjør det samme uten å lage linjeskift etterpå
- hvis du vil unngå mellomrommene, kan du bygge en string av dataene og skrive den ut:  
`print str(a) + str(b) + str(c)`
- `print` skriver også ut lister på en fin måte

# Input

- Bruker som regel `stdin.readline()`, som leser inn en linje fra terminalen som en string
- Må i så fall skrive `from sys import stdin` øverst i programmet
- `s.strip()` fjerner mellomrom på begge ender
- `s.split()` "deler opp" den originale strengen der den finner whitespace og lager en liste av delene
- `int(s)` konverterer en string til et heltall; `float(s)` konverterer til et flyttall

# if-setninger

- Fungerer på samme måte som i Java, unntatt...
  - Trenger ikke parenteser rundt uttrykkene
  - `elif` i stedet for `else if`
- Støtter sammenligninger av mer enn to operander på en gang
  - Java:  
`if (a < b && b == c && c <= d && d < e)`
  - Python:  
`if a < b == c <= d < e:`

# if-setninger forts.

- `True` og `False` staves med stor forbokstav
- `0`, `[]`, `""` og `None` tolkes som `False`; alt annet tolkes som `True`
- Boolske operatører er annerledes
  - Java: `&&`, `||`, `!`
  - Python: `and`, `or`, `not`
  - `and` og `or` er kortsluttet, akkurat som i Java (hvis mulig, evalueres bare venstresiden)

# Funksjoner

- Funksjoner defineres med `def`
- Returtype og argumenttyper spesifiseres ikke, så man kan i utgangspunktet sende hva som helst når man bruker funksjonen (men det er sjelden lurt)

```
def printArguments(a, b):  
    print a  
    print b
```

- Hvis man ikke returnerer noe selv, returneres `None` (tilsvarer `null` i Java)

# Funksjoner forts.

- Alle parametre sendes som referanser - det er altså trygt å sende lister og store objekter
- Tall og strings er immutable (de kan ikke endres), så når man modifierer slike variabler opprettes det nye objekter. "Endringer" av slike variabler inni en funksjon vil derfor ikke synes utenfor
- Lister, dictionaries, egendefinerte objekter og det meste annet er mutable, og endringer av slike variabler inni en funksjon vil endre det originale objektet

# Lists

- Som array i Java
- 0-indekserte
- Kan inneholde hva som helst
- Opprette tom liste: `list = []`
- Opprette utfylt liste: `list = [a, b, c, ..., z]`
- Endre element: `list[index] = element`
- Hente element: `list[index]`
- Finne lengde: `len(list)`
- Koble på en annen liste: `listA.extend(listB)`

# Lists forts.

- Legge til et element bakerst: `list.append(element)`
  - Hvis `element` er en liste, vil hele listen bli satt inn som ett element
- Legge til et element på en bestemt plass (ineffektivt): `list.insert(index, element)`
- Lese og fjerne element bakerst: `list.pop()`
- Lese og fjerne element på en bestemt plass (ineffektivt): `list.pop(index)`
- Finne indeksen til første forekomst av et bestemt element (ineffektivt, og krasjer hvis elementet ikke er der):  
`list.index(element)`
- Fjerne første forekomst av et bestemt element (ineffektivt, og krasjer hvis elementet ikke er der): `list.remove(element)`

# for-løkker og iterasjon

- Vi kan kun iterere over lister, dvs. gå gjennom alle elementer i en liste
- Syntaks: `for element in list:`
- Løkken vil kjøre like mange ganger som det er elementer i listen, og hver gang vil *element* ha verdien til tilsvarende element i listen
- ```
for e in [0, 42, "hei", [1, 2, 3]]:  
    print e
```

# for-løkker og iterasjon forts.

- Hvis vi ønsker en "vanlig" `for`-løkke, kan vi bruke `range()` til å lage en liste med heltall
- Tre utgaver:
  - `range(end)`  
`[0, 1, 2, ..., end - 1]`
  - `range(start, end)`  
`[start, start + 1, ..., end - 1]`
  - `range(start, end, step)`  
`[start, start + step, ..., start + x * step]`  
slutter på den siste verdien som er mindre enn end, eller siste verdi som er større enn end hvis step er negativ

# for-løkker og iterasjon forts.

- Iterasjon over elementene i en liste gjør man altså slik:

```
list = [1, 5, 2, 8, 9]
for e in list:
    print e
```

- Du kan da ikke finne indeksene ut fra elementene!

- Iterasjon over indeksene gjør man slik:

```
list = [1, 5, 2, 8, 9]
for i in range(len(list)):
    print i, ":", list[i]
```

# continue, break og else

- `continue` hopper til neste iterasjon av den innerste `for`- eller `while`-løkka
- `break` avbryter den innerste `for`- eller `while`-løkka
- Man kan plassere en `else`-blokk etter en `for`- eller `while`-løkke
  - Den vil bli utført dersom løkka avsluttes på naturlig måte, men den vil ikke bli utført dersom løkka avsluttes med `break`

# Dictionaries

- Brukes for å knytte nøkler til verdier
- Kan ses på som et array der indeksene kan være hva som helst (som er immutable)
- Implementert som hashmaps
  - Består egentlig av et array. Ut fra nøkkelen beregnes plasseringen i arrayet. Virkemåten til hashmaps dekkes senere i faget.

# Dictionaries forts.

- Opprette tomt dictionary: `dict = {}`
- Opprette utfylt dictionary: `dict = {key1 : value1, key2 : value2, ...}`
- Lagring av verdi: `dict[key] = value`
- Henting av verdi: `dict[key]` (vil krasje om `key` ikke finnes)
- Sjekke om nøkkel finnes: `dict.has_key(key)`
- Iterere gjennom verdier: `for v in dict.values():`
- Iterere gjennom nøkler: `for k in dict.keys():` eller bare `for k in dict:`

# Tuples

- En slags list som ikke kan endres etter at det er opprettet
- Kjekt for å returnere mer enn én verdi fra en funksjon

- ```
def minAndMax(list):  
    min = list[0]  
    max = list[0]  
    for x in list:  
        if x < min:  
            min = x  
        if x > max:  
            max = x  
    return (min, max)
```

# Strings

- Rammes inn med " eller ' (ingen forskjell)
- Tegn kan escapes på samme måte som i Java (\", \', \n, \t osv.)
- Strings er immutable, så alle "endringer" oppretter i virkeligheten en ny string
- Tjuvtriks for å bygge en string effektivt:  
".join(['abc', 'def'])

# Bruke array som stack

- bruk `append(element)` for å pushe på stacken (legger element til på slutten)
- bruk `pop()` for å poppe fra stacken (fjerner og returnerer elementet på slutten)

# Bruke array som queue

- `pop(0)` fjerner det første elementet
- Men dette er ineffektivt, fordi alle de andre elementene må flyttes ett hakk frem

# Listebehandling

- Sortere stigende: `list.sort()` - bruk dette gjerne til testing, men *ikke* bruk det i praksisøvingene!
- Reversere: `list.reverse()`
- Hente ut en delliste: `list[start : end]`
  - end-elementet blir ikke med
  - `list[: end]` tilsvarer `list[0 : end]`
  - `list[start :]` tilsvarer `list[start : len(list)]`
  - `list[:]` (og alle varianter (over)) gir en kopi av lista!
  - `list[i : i + 1]` lager en liste som bare består av element *i*

# Listebehandling forts.

- Hvis man bruker negative tall i  $[:]$ -notasjonen angir man da antall elementer fra slutten av lista
- $a[-x : -y]$  gir fra og med  $x$ te siste element til, men ikke med,  $y$ ende siste element
- Disse kan godt blandes, f.eks. som  $a[x : -x]$ , som gir hele lista minus de  $x$  første og de  $x$  siste

# Listebehandling forts.

- Elegant generering av lister:

```
list = [listElement for element in list if condition]
```

- Tilsvarende kode:

```
resultingList = []  
for element in list:  
    if condition:  
        resultingList.append(resultingElement)
```

- Eksempel:

```
def squareRoots(list):  
    return [sqrt(x) for x in list if x >= 0]
```

# Klasser

- `class classname:`  
    *variable1 = startValue1*  
    *variable2 = startValue2*  
    ...  
    **def** *methodA*(**self**, *arg*, ...) *...*  
    ...  
    **def** *methodB*(**self**, *arg*, ...) *...*  
    ...
- Variablene trenger ikke å listes opp på forhånd (men det er standard å gjøre det)
- Vi kommer til å bruke klasser kun for å lagre data, ikke for å gjøre fancy objektorientering

# Klasser forts.

- Python sitt svar på `this` er `self`
  - `self` må listes som første parameter i metoder
  - `self` må alltid brukes når man skal ha tak i objektvariabler
- Constructoren heter alltid `__init__`  
(to understreker på hver side)
- Til fysmat og de andre som har lært C++: Python, i likhet med Java, bruker pekere bak kulissene, men eksponerer dem ikke for programmererne. Det finnes derfor ikke noe som tilsvarer C++-operatorene `&` og `*`, og `.` brukes i stedet for `->`

# Eksempelklasse

```
class Kubbe:
    vekt = None
    neste = None
    def __init__(self, vekt):
        self.vekt = vekt
        self.neste = None

    def hentNeste(self):

        return self.neste
```

#bruk:

```
k = Kubbe(12)
```

```
m = Kubbe(8)
```

```
k.neste = m
```

```
n = k.neste           #evt. neste linje, for å illustrere klasse-funksjon
```

```
n = k.hentNeste()    #k sendes automatisk som parameteren self
```

# Eksempelklasse

Tilsvarende klasse i Java:

```
public class Kubbe {  
    int vekt;  
    Kubbe neste;  
    public Kubbe(int vekt) {  
        this.vekt = vekt;  
        this.neste = null;  
    }  
  
    public Kubbe hentNeste() {  
        return neste;  
    }  
}
```

# Ressurser

- <http://www.python.org> - Python's hjemmeside
- <http://hetland.org/python/instant-hacking.php>  
- Magnus Lie Hetland sin Python-tutorial
- <http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/>
  - Videoforelesninger fra det tilsvarende faget på MIT
  - Holdt av bl.a. Leiserson, en av forfatterene av boka
  - Det ryktes at de er mye bedre enn øvingsforelesningene