



Introduktion til programmering

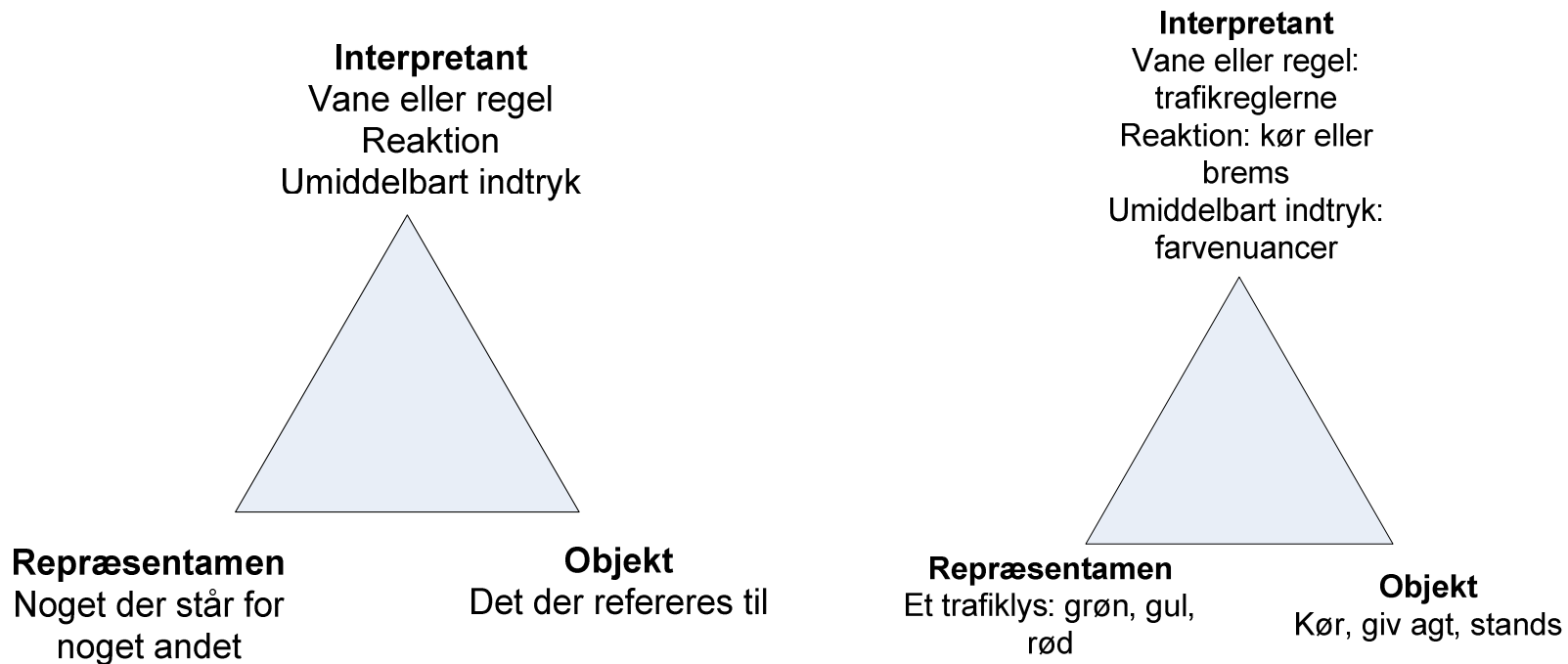
Afslutning

[Hovedideen

]

[Peirce's tegnbegreb]

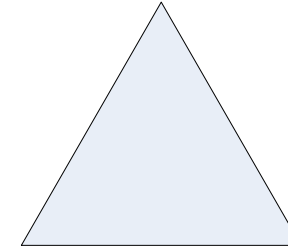
- Repræsenteramen, objekt og interpretant



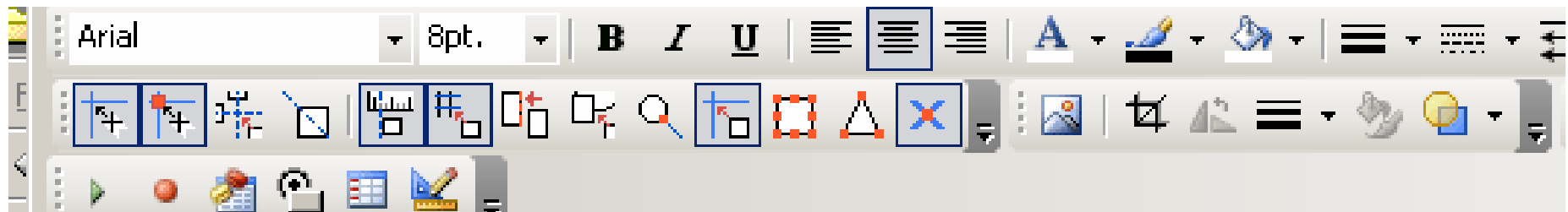
[Computerbaserede tegn]

- Interfacetegn: angiver hvad man har gjort, kan gøre, og ikke kan gøre
- Modalitet: er tilfældet, er muligt, er umuligt

Interpretant
Vane eller regel:
interfacestandard
Reaktion: klik eller
ikke klik
Umiddelbart indtryk:
farvenuancerne

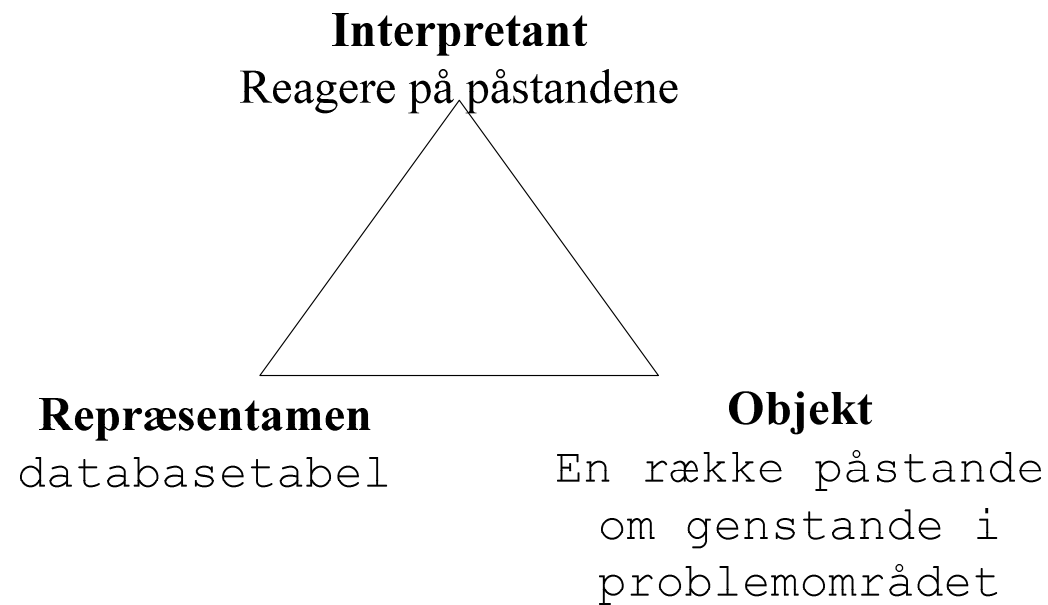


Repræsentamen	Objekt: operationen er
+ Kant, +farver, ———	Valgt
- kant, + farver ———	Kan vælges
- kant, - farver ———	Kan ikke vælges



[Databaser]

firstname	lastname	address	city	gender	email	phone	insurance	hasFather	hasMother	hasDoctor
Jeppe Boegh	Andersen	Thorsgade 20	8410	dreng		86379790	Baltica	111111112	111111113	111111114
Lotte Boegh	Andersen	Thorsgade 20	8410	pige		86379790	Baltica	111111112	111111113	111111114
Stine	Jacobsen	Thorsgade 20	8410	pige		86379790	baltica		111111116	111111114



[OO modelling]

```
■ class Person:  
■     def __init__(self,name,address, cpr):  
■         self.name = name  
■         self.address = address  
■         self.cpr = cpr  
■     def changeName(self, newName):  
■         self.name = newName  
■     def changeAddress(self,newCpr):  
■         self.cpr = newCpr  
■     def printPerson(self):  
■         print 'cpr = '+ self.cpr  
■         print 'name = '+ self.name  
■         print 'address = '+ self.address
```

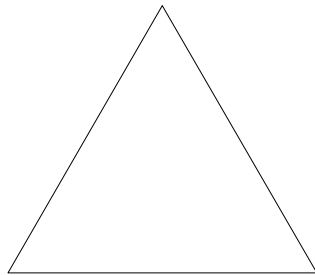
```
■ class Patient(Person):  
■     def __init__(self,name,address,cpr):  
■         Person.__init__(self,name,address, cpr)  
■         self.ward = "  
■         self.hospital = "  
■     def admit(self,ward, hospital):  
■         if self.ward == "  
■             self.ward = ward  
■             self.hospital = hospital  
■         else:  
■             print self.name + ' is already admitted'  
■     def discharge(self):  
■         if self.ward != "  
■             self.ward = "  
■             self.hospital = "  
■         else:  
■             print self.name + ' is not admitted yet'  
■     def printPatient(self):  
■         self.printPerson()  
■         print 'ward = ' + self.ward  
■         print 'hospital = ' + self.hospital
```

[OO Modelling]

- >>> from semiotik import *
- >>> patient = Patient('Karl','Vestergade 10','3112560091')
- >>> patient.admit('P5','Skejby')
- >>> patient.printPatient()
- cpr = 3112560091
- name = Karl
- address = Vestergade 10
- ward = P5
- hospital = Skejby
- >>> patient.admit('P6','Skejby')
- Karl is already admitted
- >>> patient.discharge()
- >>> patient.admit('P6','Skejby')
- >>> patient.printPatient()
- cpr = 3112560091
- name = Karl
- address = Vestergade 10
- ward = P6
- hospital = Skejby

[OO Modellering]

Interpretant
Holde systemet ajour
med problemområdet



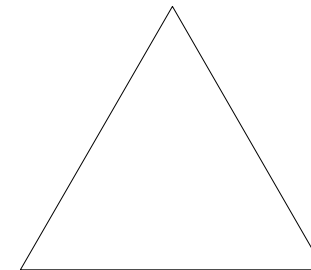
Repræsantamen

Objekt

Objekt	_____	Genstande
Funktioner	_____	Handlinger
Variable	_____	Tilstande

i problemområdet

Interpretant
Holde systemet ajour
med problemområdet



Repræsantamen

Objekt

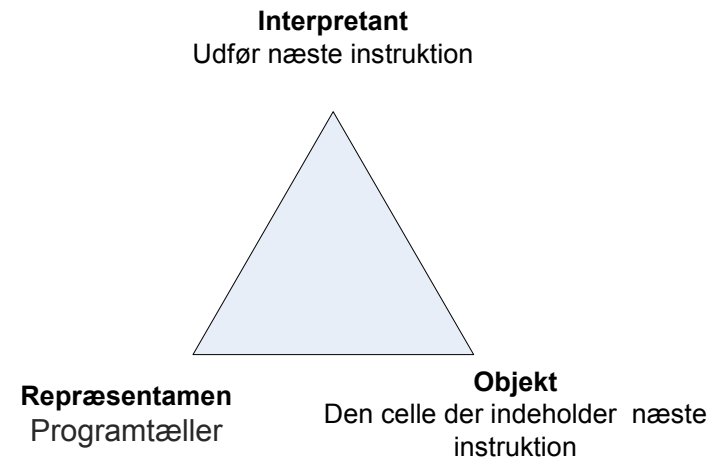
Patient	_____	En patient
Admit	_____	Indskrive patient
Ward	_____	Den afdeling der har ansvaret for patienten

[Maskinsprog

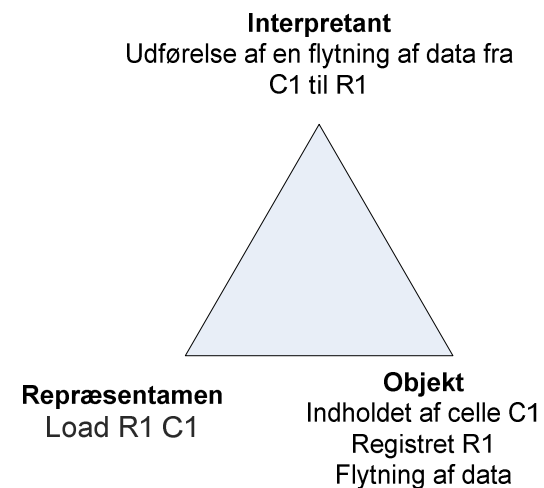
]

[Maskinel tegndannelse]

- Et program kan referere til sig selv



- eller dele af den fysiske maskine



[Databaser

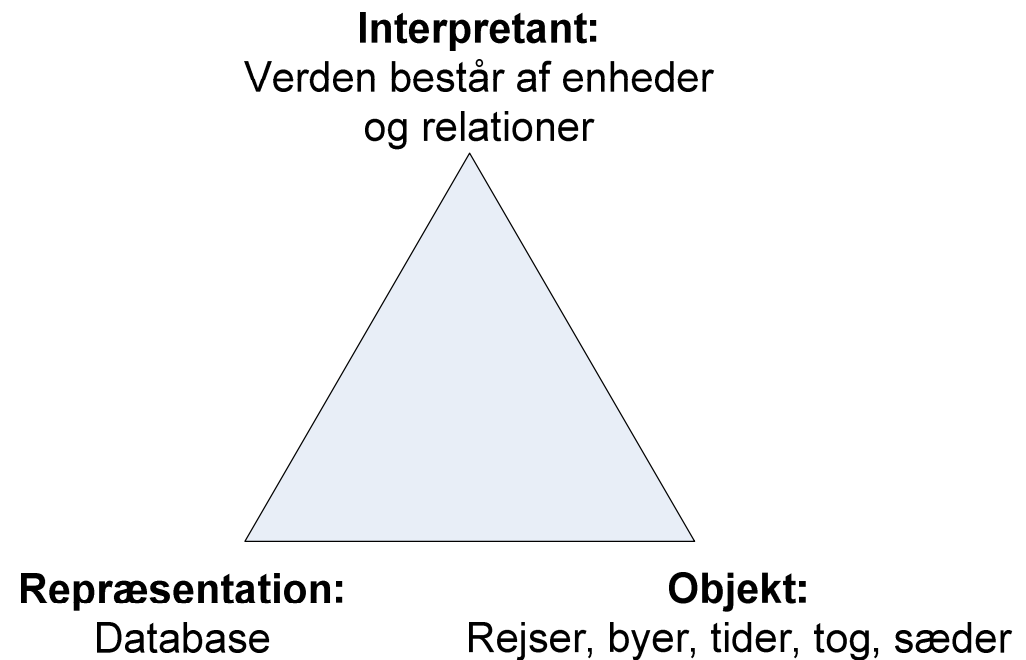
]

Databaser formidler en bestemt fortolkning

- Verden består af enheder og relationer

It's useful to view a database as
a set of propositions ...
concerning some enterprise of
which the database is supposed
to provide some kind of record

(Hugh Darwen)



[Meningsfulde relationer]

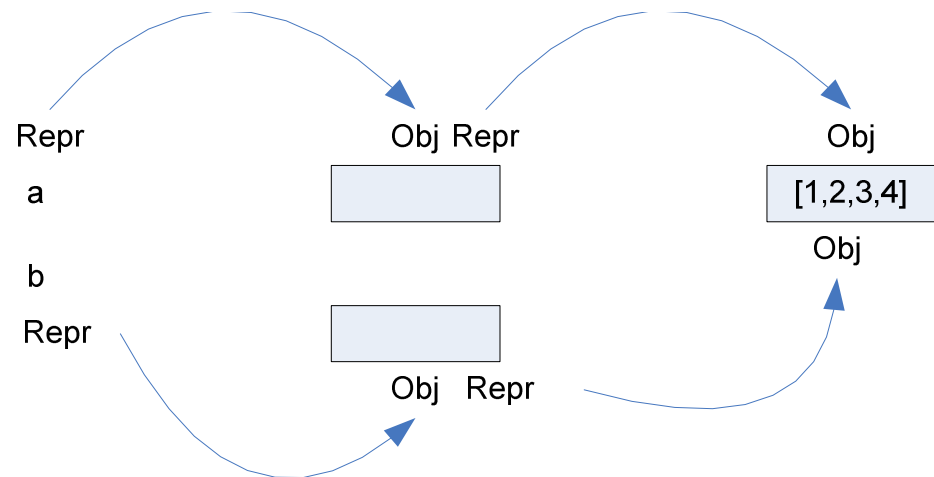
- Hvis en tabel kan parafraseres ved en en eller flere simple sætninger
 - med primærnøglen som subjekt og
 - tabelnavnet som verbum og kolonnenavnene som sætningsled eller
 - kolonne navnene som del af prædikatet
- kan vi afgøre hvad databasen repræsenterer i virkeligheden
- Ejer(personId, kæledyrsId)
 - personId ejer kæledyrsId
- Person(cpr,navn,adresse,ægtefælle)
 - Cpr hedder navn
 - Cpr bor på adresse
 - Cpr er gift med ægtefælle
- Hvis vi ikke kan omdanne tabellen til sætninger, kan vi ikke tale om dens indhold. Tabellen kan derfor ikke gå videre i organisationens kommunikation (Luhmann)

[Python

]

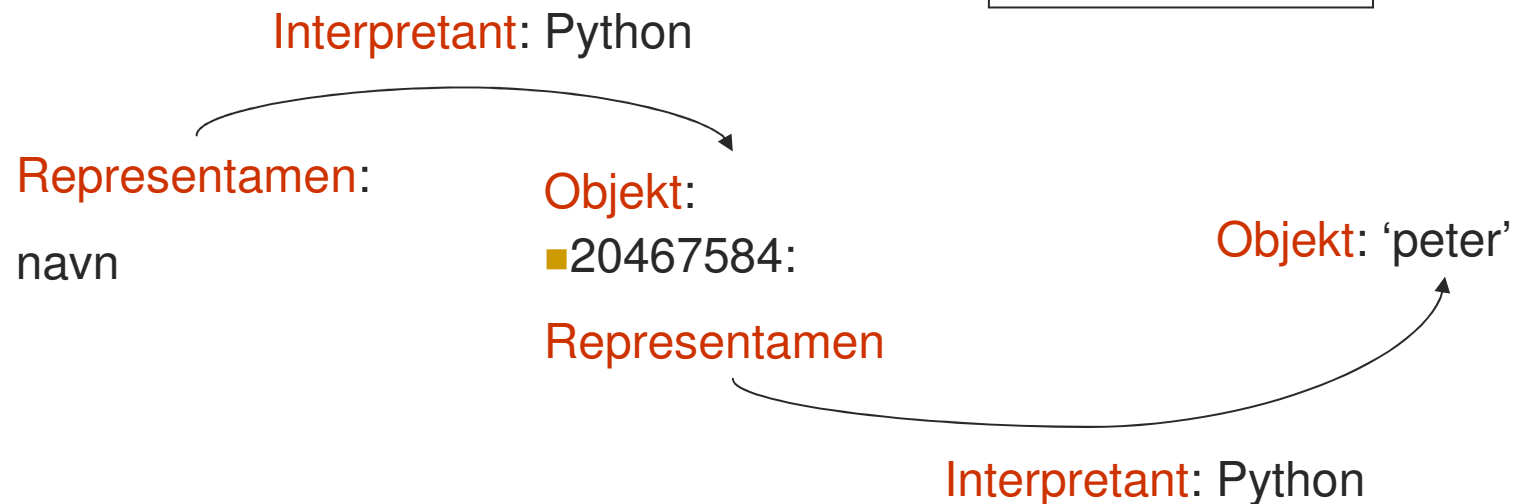
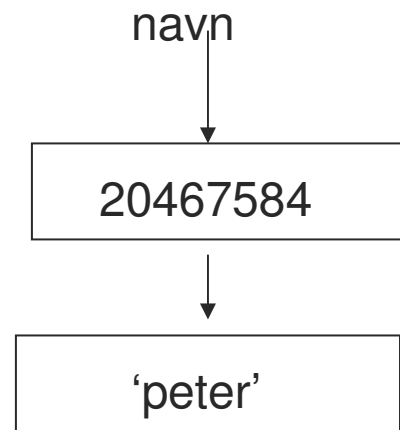
Python's udtryk repræsenterer objekter

- `a = [1,2,3,4]`
- `b = a`
- Indirekte reference.



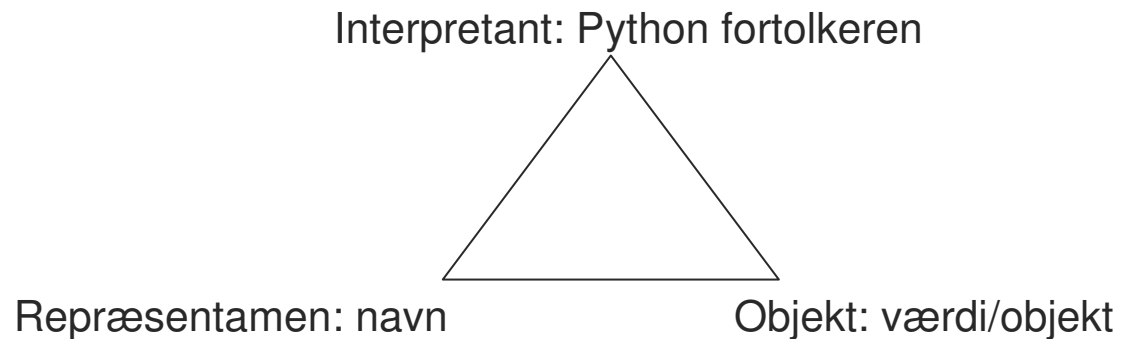
[Indirekte reference]

- `>>> navn`
- `'peter'`
- `>>> id(navn)`
- `20467584`



[Værdier]

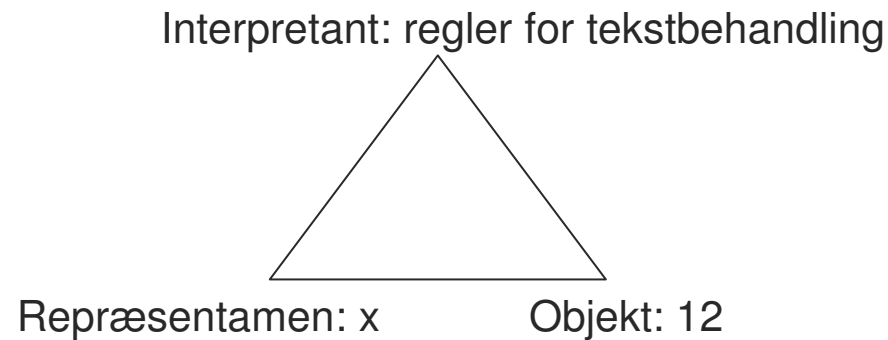
- Det grundlæggende element i Python er **værdier** der tilhører bestemte **typer**
- Værdier er næsten altid **objekter**
- Eksempler på typer:
 - Et tal eller en tekst (string)
- Nogle typer er indbyggede, andre er defineret af programmøren
- Tal og tekst er to grundlæggende indbyggede typer
- Vi refererer til værdier ved hjælp af **navne** (variable)



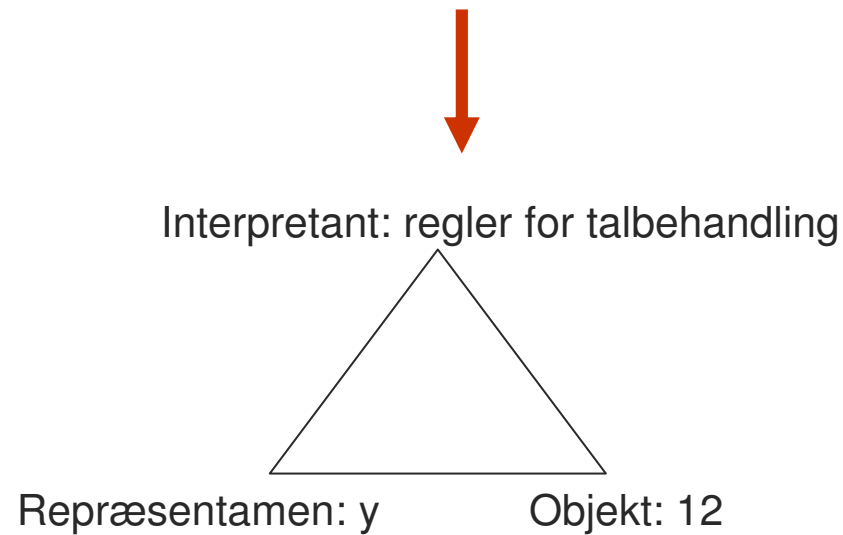
[Værdier og typer]

- `>>> navn = 'peter'` Type: tekst (string), markeret ved apostrofferne
- `>>> telefon = 86379790` Type: heltal
- `>>> navn`
 - `'peter'`
- `>>> telefon`
 - `86379790`
- `>>> type(navn)` Vi kan checke hvilken type værdien er ved hjælp af "type"
 - `<type 'str'>`
- `>>> type(telefon)`
 - `<type 'int'>`

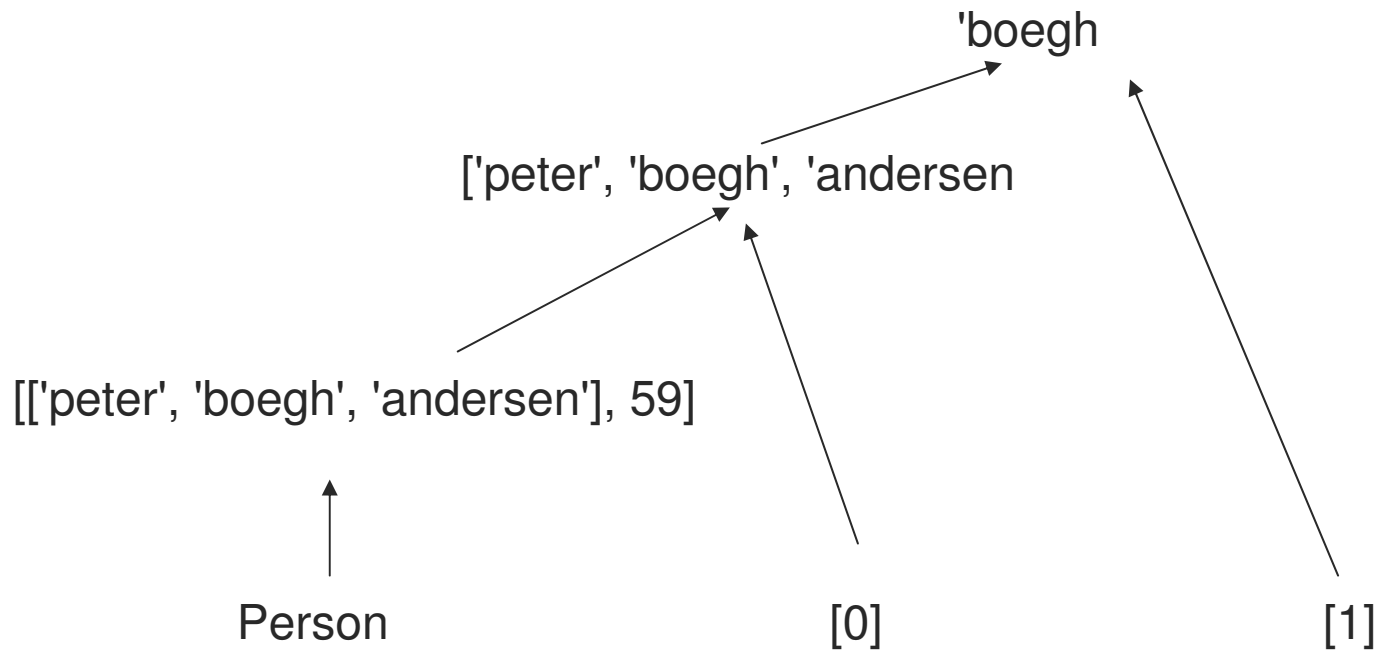
Datatypes er tolkninger af bitmønstre og kan ændres



```
>>> x = "12"  
>>> y = int(x)
```



Alle udtryk har tilknyttet værdier



Funktionsnavne repræsenterer også objekter

■ `def` udskrivOmvendt(`tekst`):

○ `return` `tekst[::-1]`

Funktionens navn

Parametrene

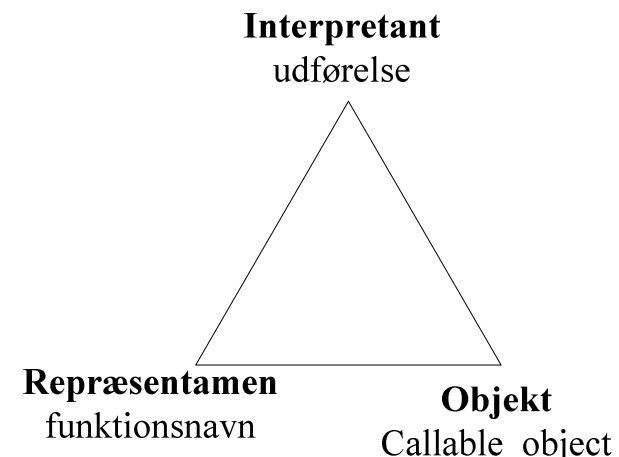
Beskrivelse af selve funktionen (callable object)

■ `>>>` `X =`

`udskrivOmvendt('peter')`

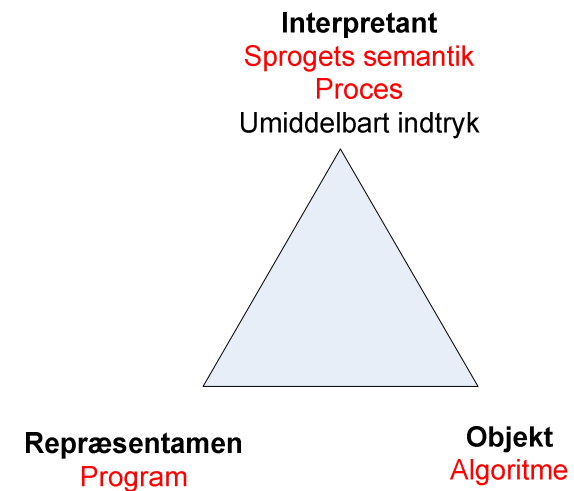
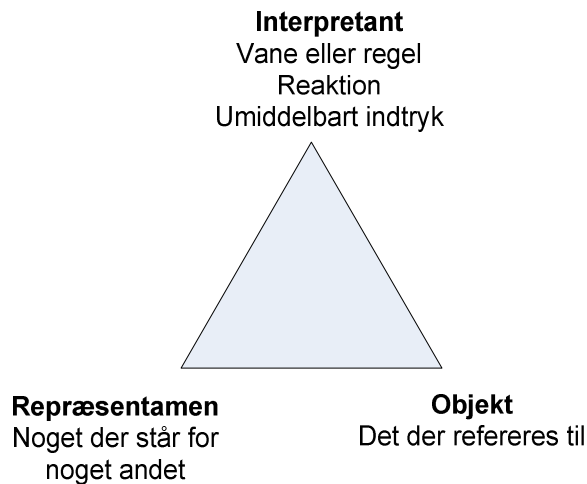
■ `>>>` `X`

○ `'retep'`



Algoritmer repræsenteres af sætninger i en programtekst

- **Program**: repræsentation af (eller flere) algoritmer
- **Proces**: udførelse af en algoritme



Repræsentationer og deres objekter

- Store dele af Python kan forstås gennem følgende simple model:
- En repræsentation/udtryk står for et objekt/en værdi
- Fortolkningen af et udtryk producerer det objekt som repræsentationen står for
- Kombinationen af en funktion/operation plus andre værdier producerer en nye værdi
- Et funktionsnavn står for et funktionsobjekt
 - `print intersection`
 - `<function intersection at 0x011851B0>`

Repræsentation	Objekt, Værdi	Eksempel
Simpelt udtryk	Liste, tekst, tupel, boolsk værdi, dictionary, heltal	[1,2,3], 'peter', (2,3,4), True
Operationer	Operation	[1:3], and, +
Sammensatte udtryk	Værdien af udtrykket	<code>str((12+34))[1] == '6'</code> , <code>1 > 2 == False</code>
Funktionsnavn	funktionsobjekt	<code>intersection</code> repræsenterer <code>0x011851B0</code> der igen repræsenterer et funktionsobjekt
Funktionskald	Værdien af det der returneres	<code>intersection([1,2,3],[2])</code> repræsenterer en liste
Sætning	tilstandsforandringer	

[Menneskesprog og
programmeringssprog

]

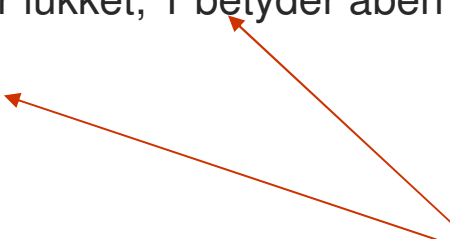
Samme sag kan udtrykkes i forskellige sprogformer 1

- Dansk
- Så længe der er muslinger i spanden:
 - Tag en musling op af spanden
 - Hvis muslingen er lukket så:
 - Smid den i gryden
 - Ellers:
 - Smid den i affaldsposen
- Kog muslingerne i 5 min
- Så længe der er muslinger i gryden
 - Tag en musling op af gryden
 - Hvis muslingen er åben så:
 - Læg den på fadet
 - Ellers:
 - Smid den i skraldespanden

Samme sag kan udtrykkes i forskellige sprogformer 2

- Python
- # Spanden indeholder muslinger. 0 betyder lukket, 1 betyder åben
- spand = [1,0,0,1,0,1,0,0]
- # gryde og affaldspose er tomme i starten
- gryde = []
- affaldspose = []
- # så længe der er muslinger i spanden
- while spand <> []:
 - # tag en musling op
 - enmusling = spand[0]
 - del spand[0]
 - # hvis muslingen er lukket
 - if enmusling == 0:
 - #så smide den i gryden
 - gryde.append(enmusling)
 - # ellers
 - else:
 - #smid den i affaldsposen
 - affaldspose.append(enmusling)

Kommentarer til menneskelig fortolkning

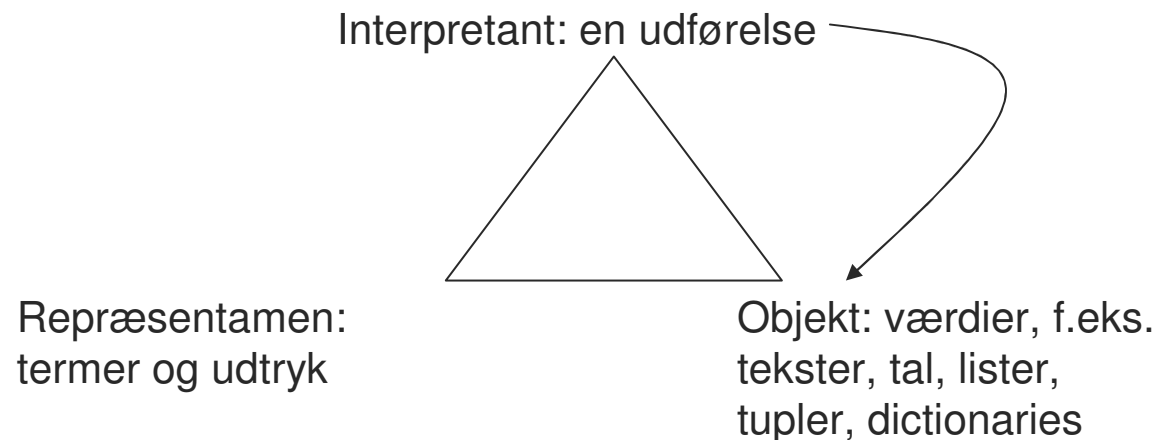


[Sammenligning]

- Samme algoritme, forskellige programmer
- Menneskelig versus maskinel fortolker
 - Computerprogrammer skal læses af både mennesker og maskiner
- Interpretanten er under forhandling ved menneskelige fortolkere, ikke ved maskinelle
 - Man laver tonsvis af fejl i starten!!
 - Naturlie sprog udvikler sig kontinuerligt
 - Computersprog er stabile i perioder og udvikler sig i versioner (Python 2.3 → Python 2.4)
- Abstraktion
 - Muslinger er repræsenteret ved én egenskab: lukket eller åben

[En væsentlig forskel]

- Udtryk er repræsentationer
- Deres objekt er værdier
- De frembringer de værdier de repræsenterer
- Det gør de naturlige sprog ikke
 - At sige "en Peugeot 307" frembringer ingen bil
 - Men at skrive 'Peter'[1:3] frembringer teksten 'et'

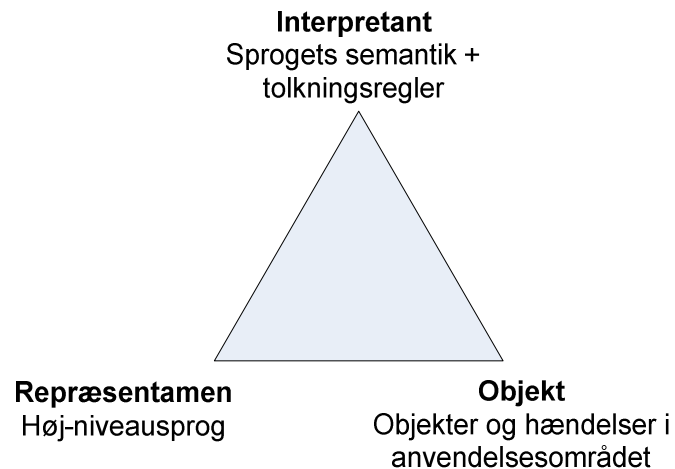


[Tolkingsregler er nødvendige]

- Muslingeprogrammet kan tolkes som udsagn om hvordan man tillaver muslinger givet bestemte tolkningsregler
- `spand = [1,0,0,1,0,1,0,0]` #spand = en spand med muslinger
- `gryde = []` #den gryde vi vil kode muslingerne i
- `affaldspose = []` #den affaldspose vi smider de dårlige muslinger i
- `while spand <> []:` # spanden er ikke tom
 - `enmusling = spand[0]` # tag en musling op
 - `del spand[0]`
 - `if enmusling == 0:` #muslingen er lukket
 - `gryde.append(enmusling)` #læg den i gryden
 - `else:`
 - `affaldspose.append(enmusling)` #læg den i affaldspose

[Tolkningsreglerne]

- Tolkningsregler
 - En **liste** tolkes som en **kontainer**
 - EN **tom** liste tolkes som en **tom** kontainer
 - **0,1** tolkes som **muslinger i en bestemt tilstand**
 - **Enmusling** tolkes som den **musling** jeg har i hånden
 - **Slettelse** af et element fra en liste tolkes som det at **tage noget** op af en kontainer
 - **Tilføjelse** af et element til en liste tolkes som det at **lægge noget ned** i kontaineren.



Scope = Python's regler for tolkning af navne

- Scope = samtalekontekst
- Namespace = interpretant der angiver hvad navne står for
 - In Python: en slags dictionary hvor navne er knyttet til objekter
 - {'A':integerobjekt, 'B': funktionsobjekt,...}
- Namespaces er knyttet til kontekster – lige som i naturlige sprog

[Hvad betyder et navn?]

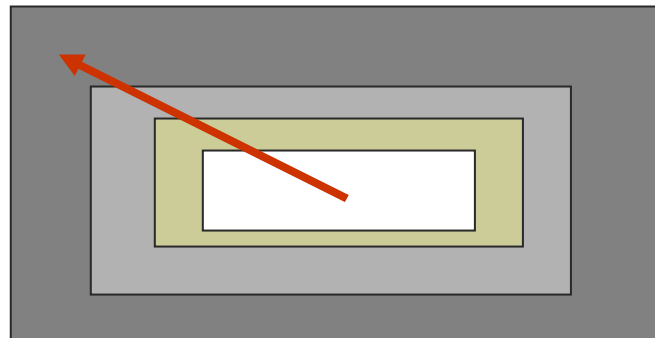
- Samtalekonteksten og tidsforløbet afgør hvad en repræsentation (et navn) repræsenterer.
- **Det danske samfund:**
 - "Anders Fogh" betegner statsministeren
 - Vi kan bruge "Fogh" som henvisning til statsministeren: "Fogh" → statsministeren
 - **Familiesammenkomst (del af det danske samfund):**
 - Hvis intet andet er sagt gælder reglen stadig. "Fogh har en del bøvl med sine ministre i øjeblikket"
 - Men hvis der kommer en replik
 - "Kan I huske Købmand Fogh"
 - "Ja, ja"
 - "Fogh har lige solgt sin forretning"
 - Så etableres referencen
 - "Fogh" → købmanden

[I naturlige sprog]

- Referencen afgøres af
 - Egenskaber ved objektet (han/hunkøn, andre egenskaber)
 - Hvornår objektet sidste blev omtalt og hvor vigtigt det er
 - Om der er nævnt andre kvalificerede objekter imellem taletidspunktet og sidste omtale
- Eksempel
 - "Prøv at se den mand derovre
 - "Ja, han (**manden derovre**) er noget nervøs"
 - "Men se så ham bagved"
 - "Han (**manden bagved**) ser mere ud til at have kontrol over tingene".

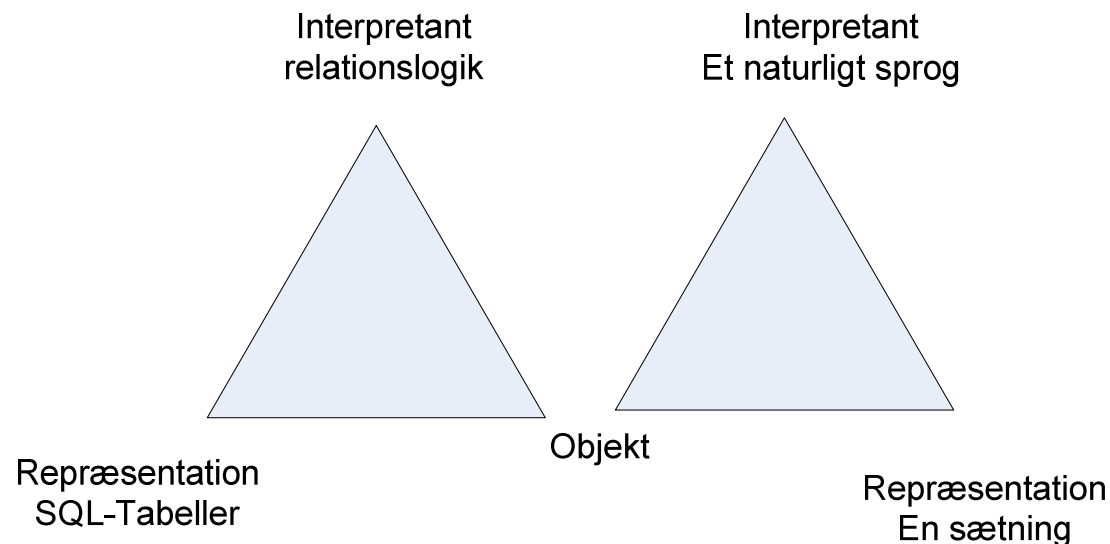
[Scope i Python]

- Lokale navne bruges i funktioner
- De forsvinder når funktionen er kørt færdig
- Der er 4 samtalekontekster
- Indbyggede navne (open, range, etc)
 - Modulet (globale): navne der er blevet tilskrevet (A = 9) en reference i modulet eller er blevet erklæret globale
 - Lokale navne i omgivende funktioner
 - Lokale navn i funktionen: navne der er blevet tilskrevet en reference i modulet



Samme sag kan udtrykkes i forskellige sprogformer

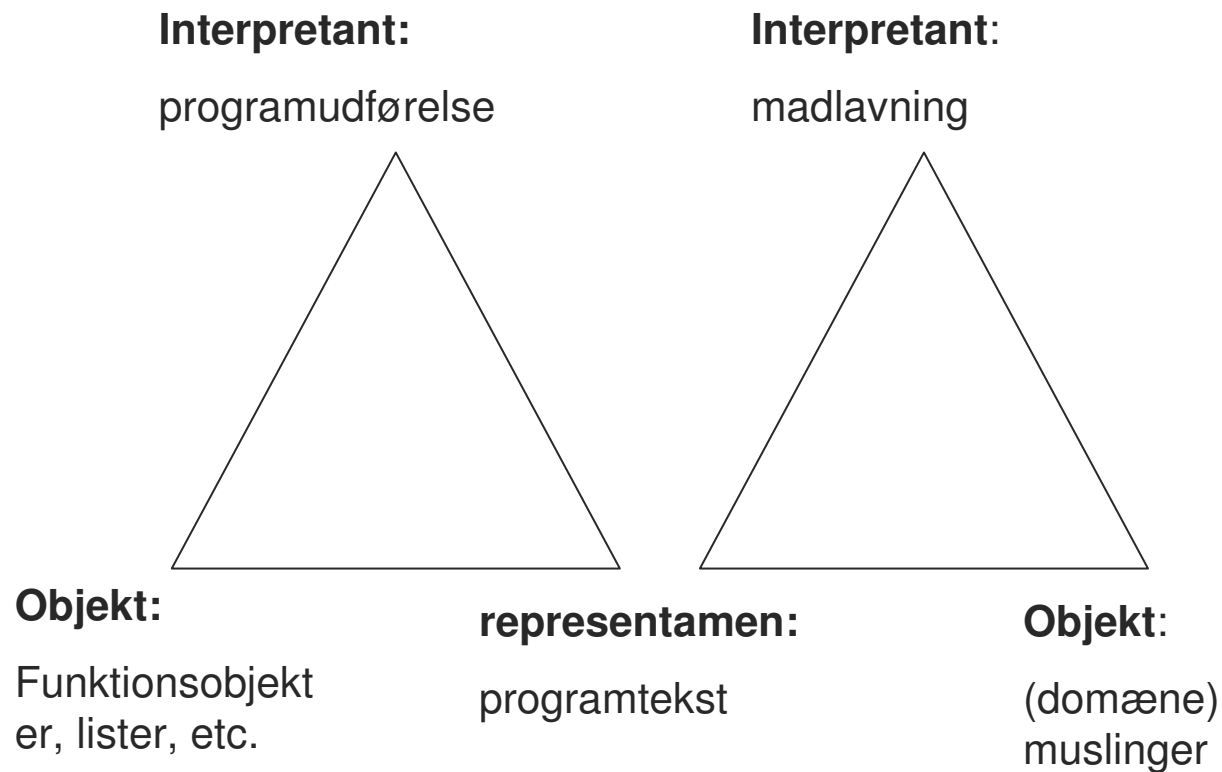
- Digitale medier er led i en kommunikationskæde der også involverer de naturlige sprog
- Der er forskel på hvad man kan udtrykke i digitale medier og naturlige sprog
- Digitale medier er typisk mere restriktive men tilbyder nye muligheder på grund af disse restriktioner.



[Samme repræsentation tolkes forskelligt af forskellige fortolkere]

- **Maskinen** tolker et program som instruktioner om at skifte tilstand
- **Programmøren** tolker et program som
 - Instruktioner til maskinen om at skifte tilstand
 - En beskrivelse af genstande og hændelser i problemområdet
- **Brugeren** tolker en programudførelse som en beskrivelse af genstande og hændelser i problemområdet

[Programtekstens to tolkninger]



[Indkapsling og fortolkningskift]

- Funktioner er også en metode til at skifte fortolkning
- Funktionens navn repræsenterer én del af verden – f.eks. problemområdet
- Funktionens indre repræsenterer en anden del af verden – f.eks. computeren.
- Repræsentation af begivenheder i computeren → Repræsentation af begivenheder i problemområdet

[Afsnit 1: maskinen]

- #en verden af lister og tal
- def KoekkenUdstyr():
- return ([],[],[])

- def TagOpAf(enKontainer):
- EtElement = enKontainer[0]
- del enKontainer[0]
- return EtElement

- def Laegl(etElement,enKontainer):
- enKontainer.append(etElement)

- def FyldMed(enKontainer, etIndhold):
- enKontainer.extend(etIndhold)

- def Toem(enKontainer):
- del(enKontainer[:])

- def DerErNogetI(enKontainer):
- return enKontainer <> []

- def ErLukket(etElement):
- return etElement == 0

- def KoekkenTilstand(spand, gryde, affaldspose):
- print ""
- spandens indhold: %s.
- Grydens indhold: %s.
- Affaldsposens indhold: %s"" % (spand, gryde, affaldspose)

[Afsnit 2: madlavning]

- #en verden af muslinger og koekkenredskaber
- (spand, gryde, affaldspose) = KoekkenUdstyr()
- FyldMed(spand, [1, 0, 0, 1, 0, 1, 0, 0])
- Toem(gryde)
- Toem(affaldspose)
- KoekkenTilstand(spand, gryde, affaldspose)
- while DerErNogetI(spand):
- enmusling = TagOpAf(spand)
- if ErLukket(enmusling):
- Laegl(enmusling, gryde)
- else:
- Laegl(enmusling, affaldspose)
- KoekkenTilstand(spand, gryde, affaldspose)

Endnu tættere på domænefortolkning

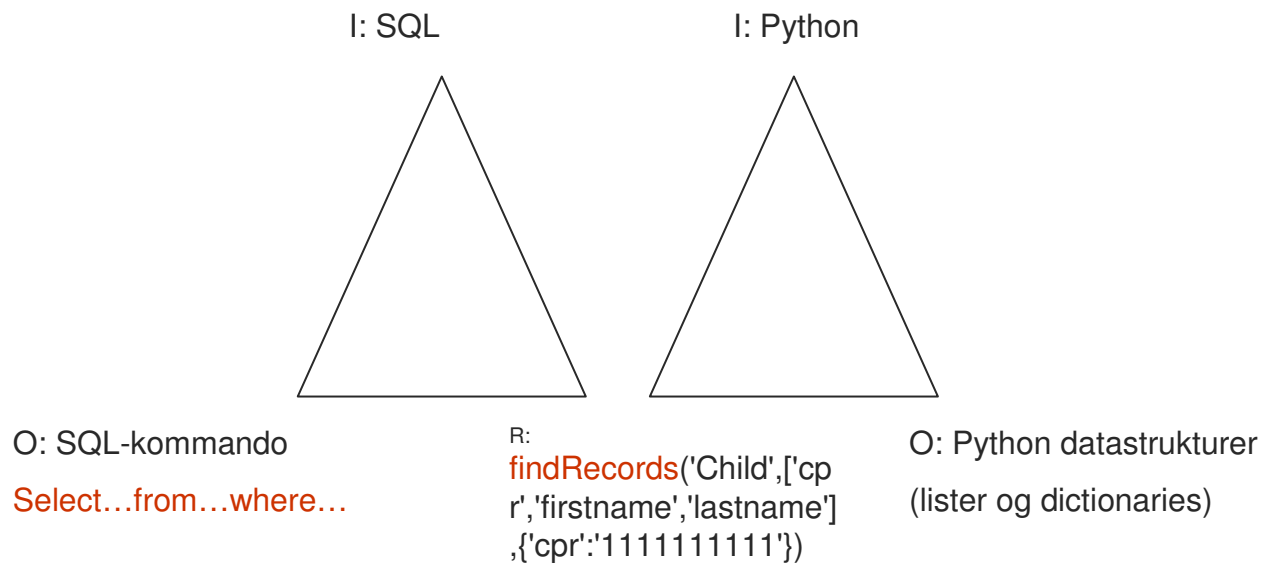
- `bucket = Container()`
- `pot = Container()`
- `wastebag = Container()`

- `you.fill(bucket, 'with', Mussels())`

- `while bucket.contains(Mussel):`
- `you.take(it, 'from', bucket)`
- `if it.isClosed():`
- `you.put(it, 'into', pot)`
- `else:`
- `you.put(it, 'into', wastebag)`

[Modulet database]

- Sørger for overgangen mellem python-verdnen og sql-verdnen

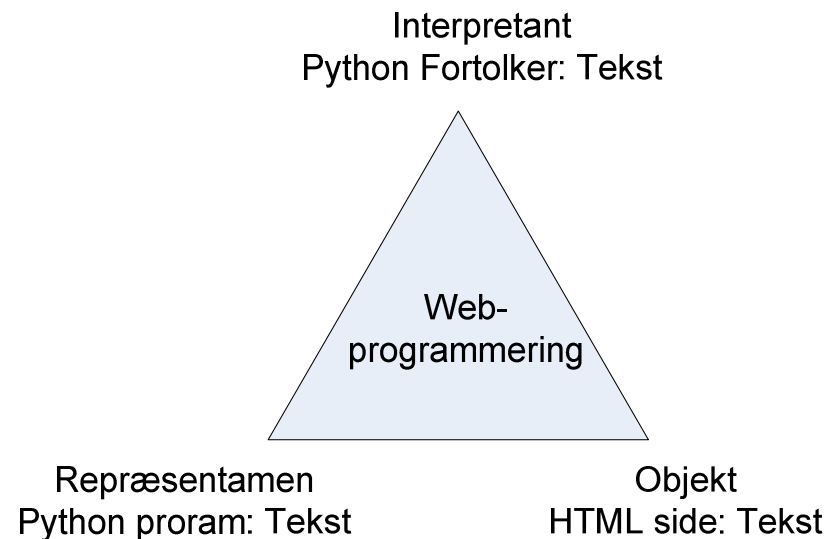


Meta-repræsentationer og selv-reference

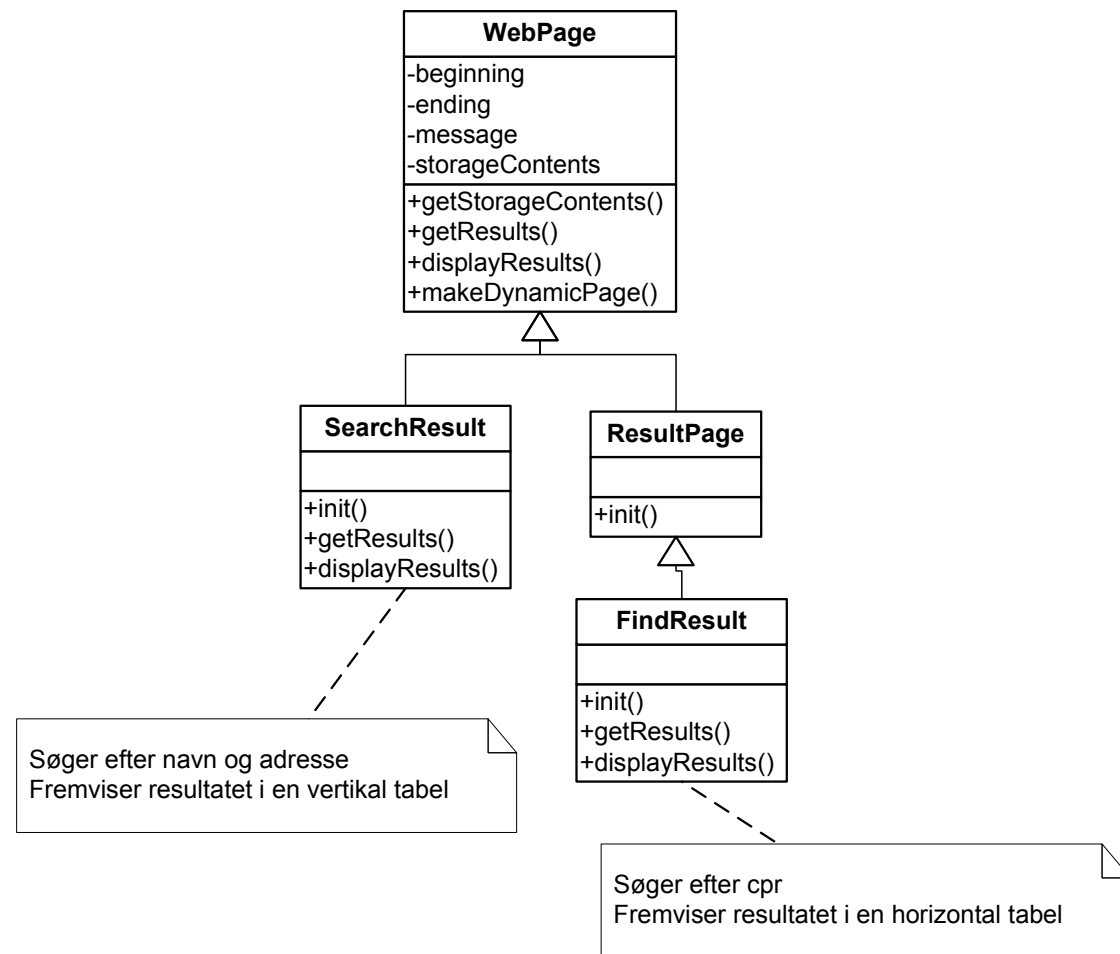
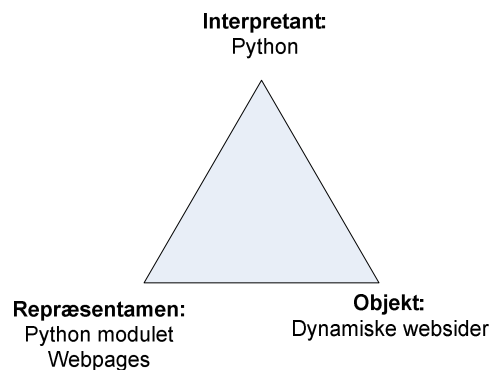
- Meta-repræsentationer: repræsentationer hvis objekter er andre repræsentationer
- Selv-reference: repræsentationer der repræsenterer sig selv

[Meta-repræsentationer]

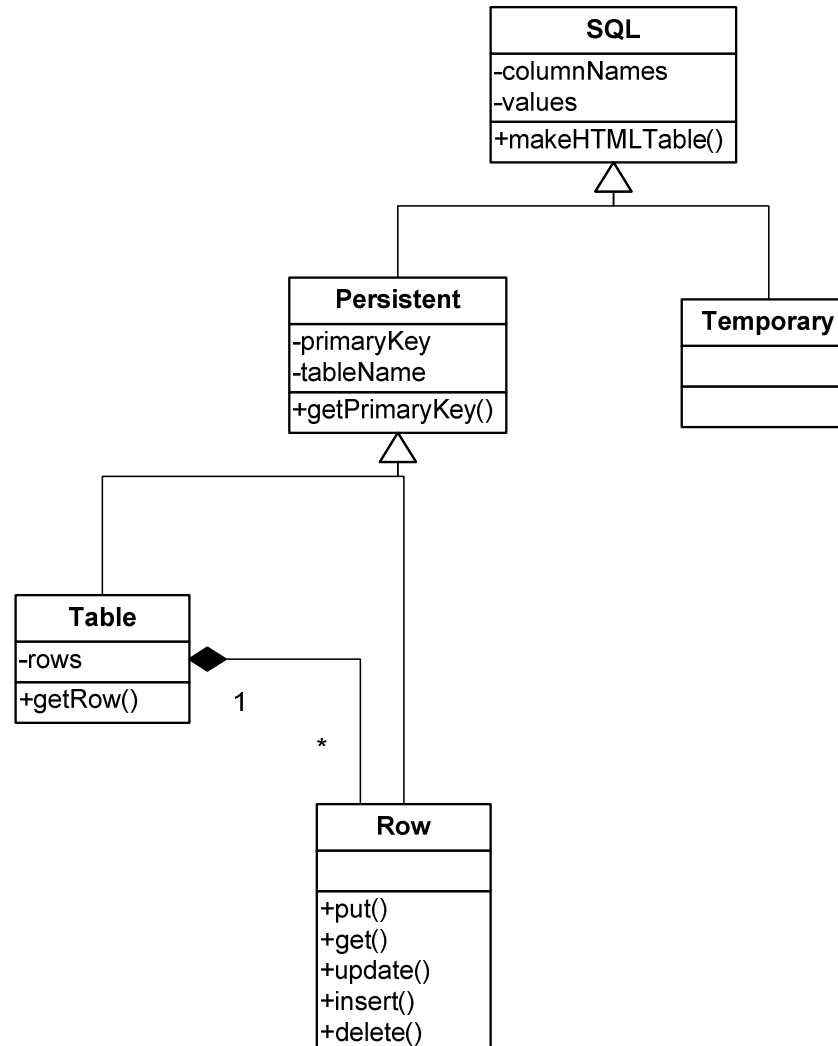
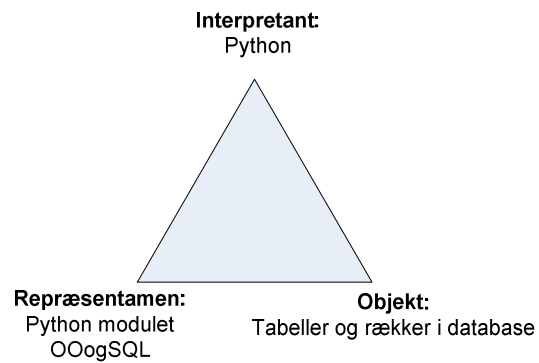
- Tekster der repræsenterer tekster ved hjælp af tekster!!!
- Web-programmering



Modulet **Webpages** repræsenterer websider



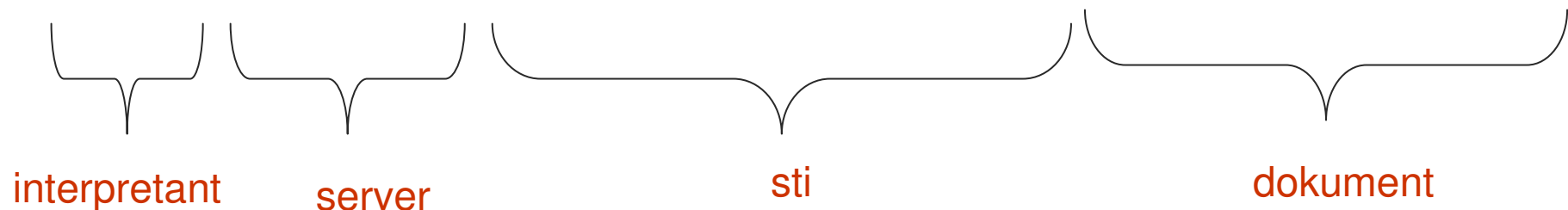
Modulet OOogSql repræsenterer en database



[Selv-reference: URL'er repræsenterer deres egen interpretant]

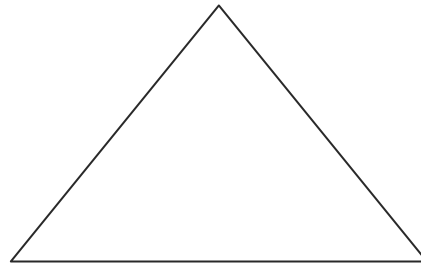
- En URL er en sammensat **repræsentation**
- Dens **interpretant** er fastlagt i HTTP-standarden
- De **objekter** den repræsenterer er:
 - En fortolkningsregel
 - En server
 - En sti på serveren
 - Et dokument

<http://imv.au.dk/~pba/Homepagematerial/programmering.htm>



[URL'er som selv-refererende tegn]

Interpretant: HTTP



repræsentamen: URL

Objekt: interpretant,
server, sti, dokument

[Selv-reference]

- URL'en repræsenterer sin egen fortolkningsregel (http)
- Nødvendigt i et dynamisk foranderligt medium hvor regler skifter hele tiden.

Praktisk eksempel på selv-reference

- Når I skal sætte Python til at lave dynamiske hjemmesider skal I sørge for to selv-referencer:
- Hvad er jeg for en slags dokument?
 - `print '''Content-type: text/html'''`
- Hvilken version af HTML og hvilket sprog er jeg skrevet i?
 - `def printOverskrift(titel):`
 - `print '''`
 - `<?xml version="1.0" encoding="iso-8859-1"?> <!DOCTYPE`
`html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "DTD/xhtml1-`
`strict.dtd"> <html xmlns="http://www.w3.org/1999/xhtml"`
`xml:lang="da">`

[Introspektion]

- Python indeholder en række funktioner der giver direkte adgang til dens indvolde
- Objekt.__dict__: en dictionary der gemmer objektets attributter og værdier
- Dir(object): returnerer alle de attributter og metoder der er knyttet til objektet.

[Introspection]

- `def introspect(self):`
- `text = 'Dictionary: '`
- `text += '\n'+ str(self.__dict__)`
- `text += '\n'+ 'Attributes and functions: '`
- `text += '\n'+ str(dir(self))`
- `return text`

Self – en grundlæggende selv-reference

- **self** henviser til det nydannede objekt
- Når man i klassedefinitionen vil henviser til objektets egne funktioner eller variable skal det ske via **self**
- ```
def getName(self):
```
- ```
    return self.name
```

Rekursion: algoritmisk selv-reference

- En rekursiv algoritme refererer til dele af sig selv

[Problematisk rekursion]

- Løsning af ægteskabelige problemer
 - Tal sammen uden at skændes
 - Forstå hinanden
 - Forstå hinanden
 - Tal sammen uden at skændes
- Løsning af økonomiske problemer
 - Skaf penge:
 - Invester i aktier
 - Invester i aktier:
 - Skaf penge
- Fører til en uendelig regres

[Effektiv rekursion]

- Fremgangsmåde
 - Basis tilfældet:
 - Involverer ikke rekursion men kun primitive operationer der umiddelbart kan udføres
 - Det rekursive tilfælde:
 - Involverer rekursion men for et simplere problem

[Rekursiv sortering]

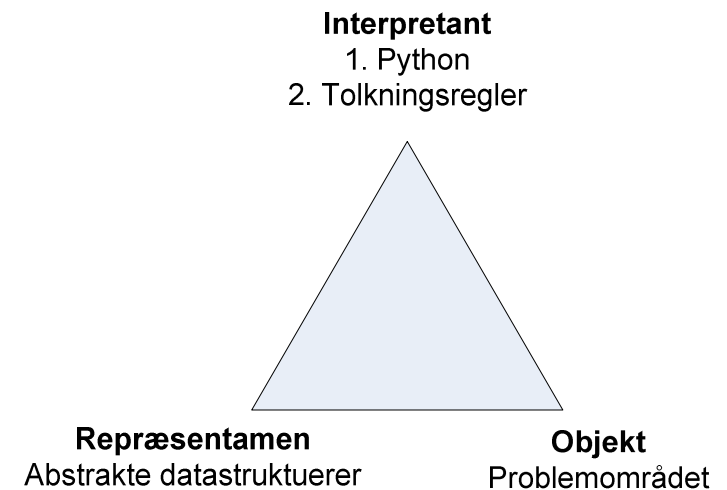
- Sortering af liste
 - if listens længde ≤ 1 : Basistilfældet: primitiv operation
 - **returner listen**
 - **else:** Rekursivt kald af et mindre kompliceret problem
 - returner det mindste element i listen +
 - **Sortering af listen der er fratrukket dette mindste element**

Tolkning i termer af problemområdet

- Datastrukturer og organisationsanalyse

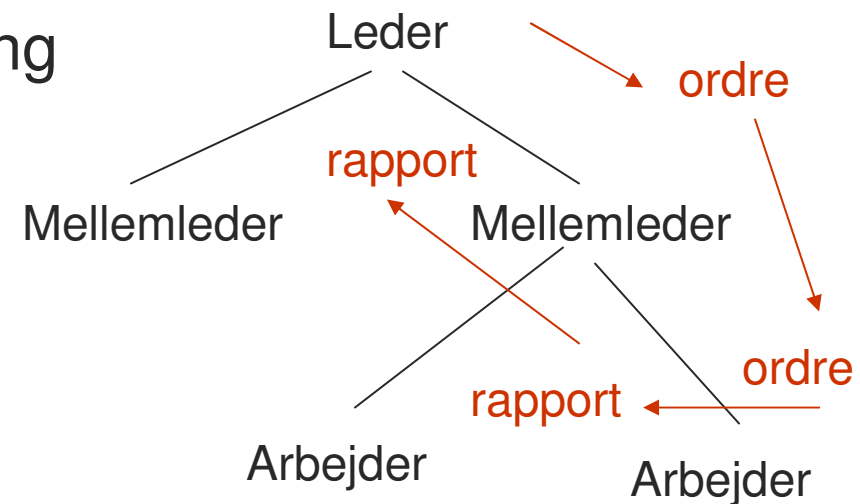
Tolkning af abstrakte datastrukturer

Repræsentation	Objekt
Lister	Associationer
Stakke	Mål og midler Prioritering og afbrud
Køer	Køer i supermarkedet Trafikpropper
Træer	Helheder og dele Overordnet og underordnet
Rekursion	Mål og midler Helhed og del



[Organisationer som maskiner]

- Bureaukratiske organisationer
 - Træer
 - Kommunikationen følger grenene på træerne
 - Køer (Dora).
 - Galt valg af tolkning

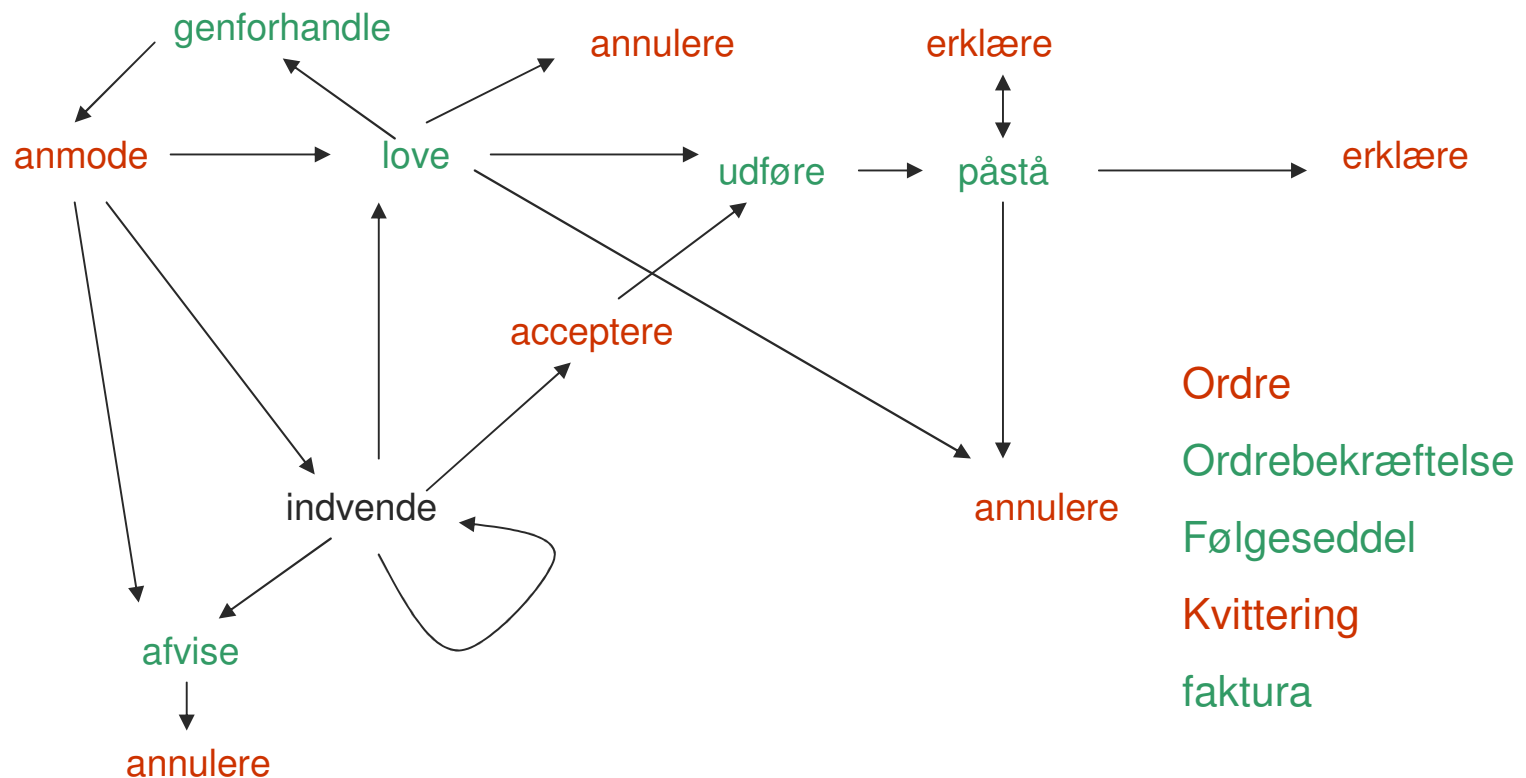


[Organisationer som flux]

- Kommunikation skaber ny kommunikation.
- Organisationen og dens grænser om- og genskabes ved kommunikation
- Connected, directed graph with loops

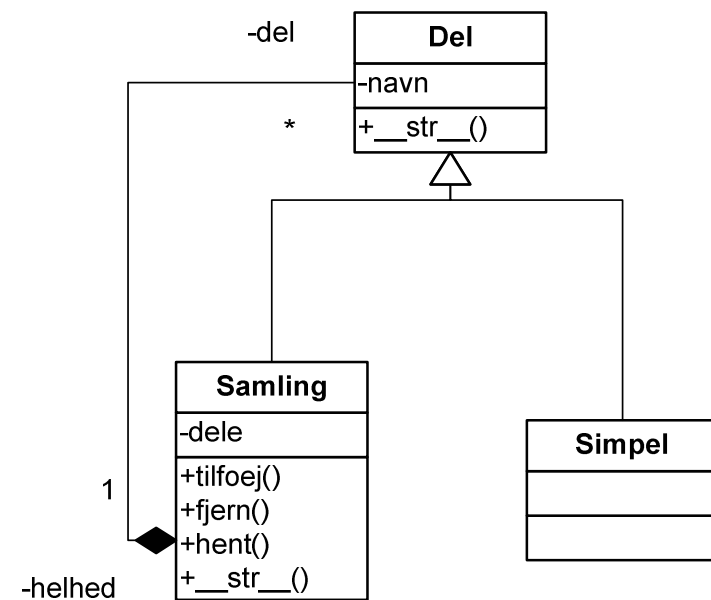
[Kommunikationsstøtte]

- Holde styr på faser og forpligtelser
 - Winograd & Flores: understanding computers and cognition, 1986.

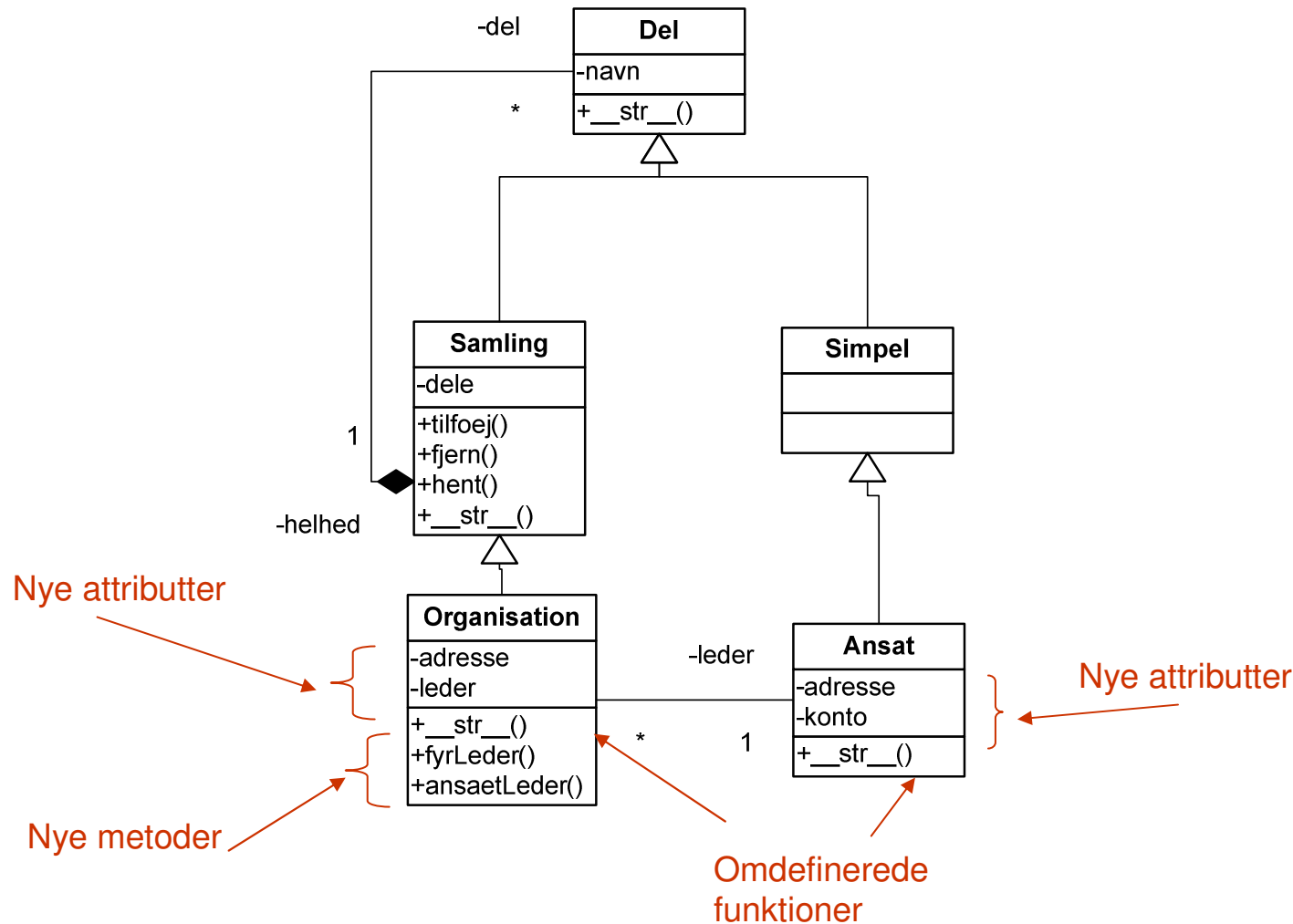


[Composite mønstret: et træ]

- Der er en én-mange relation mellem en samling og dens dele
- En relation er et abstrakt begreb der kan implementeres på mange måder.
- SQL: som en relation
- Python: som en liste af objekter



Specialisering af mønstret til beskrivelse af organisation



Vi har at gøre med et hierarki (et træ)

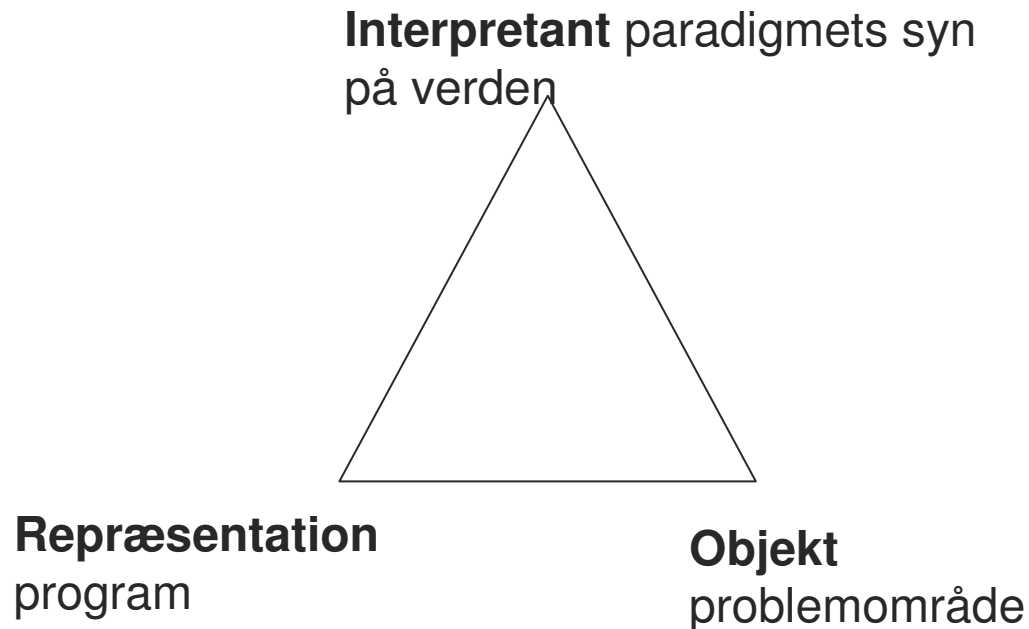
Hierarkiet er at typen organisation

[Programmeringssprog]

- Programmeringssprog udtrykker en tolkning af virkeligheden

[Programmeringsparadigmer]

- Et paradigme udtrykker et syn på problemområdet



[Imperative]

- Interpretant:
 - Verden består af aktive processer og passive ting
 - Tingene flyder mellem processerne og skifter tilstand
- Programmering
 - Udform en algoritme der omdanner ting
 - En algoritme er en sekvens af tilstandsændringer
- Ontologi
 - Subjekt → Objekt
 - Temporalt struktureret
- **Tekstgenre: narrativ**

[Objektorienterede]

- Interpretant:
 - Verden består af **aktive** objekter samt af kommunikation mellem disse objekter
 - Objekterne har metoder tilknyttet
 - `>>> x = ['a','b','c']`
 - `>>> x.extend([1,2,3])`
- Programmering:
 - lav en struktur af objekter hvis vekselvirkning resulterer i det ønskede resultat
 - Simuler problemområdet
- Ontologi
 - Subjekt/Objekt ↔ Subjekt/Objekt
- Note
 - Indeholder normalt en procedural komponent (C++, Python)
- Understøtter indkapsling og genbrug
- **Tekstgenre: deskriptiv.**

[Funktionelle]

- Interpretant
 - Verden består af afhængigheder
 - Inspireret af matematikken
- Programmering
 - Lav et hierarki af funktioner der kalder andre funktioner
 - Funktion(funktion(funktion(data)))
- Ontologi
 - Input-output, a-temporal
- Python har et funktionelt subset
 - (apply, map, reduce, listcomprehension)

[Deklarative]

- Interpretant
 - Verden består af sande eller falske sætninger samt af logiske slutningsprocesser
- Ontologi
 - Logisk positivisme
 - Logisk, a-tamporal
- **Tekstgenre: argumentativ**

[Prolog 1: påstande]

- far(peter,jeppe).
- far(peter,lotte).
- mor(else,jeppe).
- mor(else,lotte).
- far(jeppe,estrid).
- mor(lotte,erik).
- mor(lotte,ida).

- han(peter).
- han(jeppe).
- han(erik).
- hun(else).
- hun(estrid).
- hun(ida).
- hun(lotte).

[Prologg 2: slutningsregler]

- farfar(Aeldre,Yngre) :- far(Aeldre,Mellem),far(Mellem,Yngre).
- farmor(Aeldre,Yngre) :-
mor(Aeldre,Mellem),far(Mellem,Yngre).
- morfar(Aeldre,Yngre) :-
far(Aeldre,Mellem),mor(Mellem,Yngre).
- mormor(Aeldre,Yngre) :-
mor(Aeldre,Mellem),mor(Mellem,Yngre).

- soen(Yngre,Aeldre) :- han(Yngre),far(Aeldre,Yngre).
- soen(Yngre,Aeldre) :- han(Yngre),mor(Aeldre,Yngre).

- datter(Yngre,Aeldre) :- hun(Yngre),far(Aeldre,Yngre).
- datter(Yngre,Aeldre) :- hun(Yngre),mor(Aeldre,Yngre).

[Prolog 3: slutningsregler]

- `bedstemor(Aeldre,Yngre) :- mormor(Aeldre,Yngre).`
- `bedstemor(Aeldre,Yngre) :- farmor(Aeldre,Yngre).`
- `bedstefar(Aeldre,Yngre) :- morfar(Aeldre,Yngre).`
- `bedstefar(Aeldre,Yngre) :- farfar(Aeldre,Yngre).`

- `barnebarn(Yngre,Aeldre) :- bedstemor(Aeldre,Yngre).`
- `barnebarn(Yngre,Aeldre) :- bedstefar(Aeldre,Yngre).`

- `soennesoen(Yngre,Aeldre) :- soen(Yngre,Mellem), soen(Mellem,Aeldre).`
- `datterdatter(Yngre,Aeldre) :- datter(Yngre,Mellem), datter(Mellem,Aeldre).`

- `soeskende(A,B) :- far(X,A), far(X,B), not(A = B).`
- `soeskende(A,B) :- mor(X,A), mor(X,B), not(A = B).`

[Prolog 4: forespørgsler]

- 1 ?- barnebarn(X,peter).
- X = erik ;
- X = ida ;
- X = estrid ;
- No
- 2 ?- farfar(peter,X).
- X = estrid ;
- No
- 3 ?- morfar(peter,X).
- X = erik ;
- X = ida ;
- No
- 4 ?- soeskende(erik,X).
- X = ida ;

[Summa summarum]

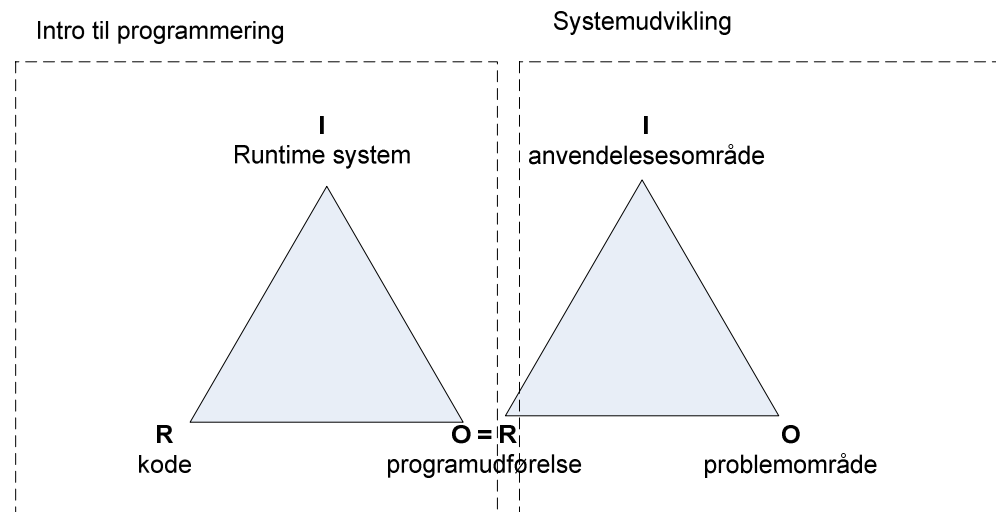
- IT-systemer er fortolkninger af virkeligheden
- De anlægger et bestemt perspektiv
- Kode har altid tre fortolkninger:
 - Hvad maskinen skal gøre når den eksekverer koden
 - Hvad udvikleren har til hensigt koden skal repræsentere i problemområdet
 - Den måde brugeren fortolker en udførelse på.
- Det gælder om at de tre fortolkninger ikke er i modstrid – men det er svært

[Systemudvikling

]

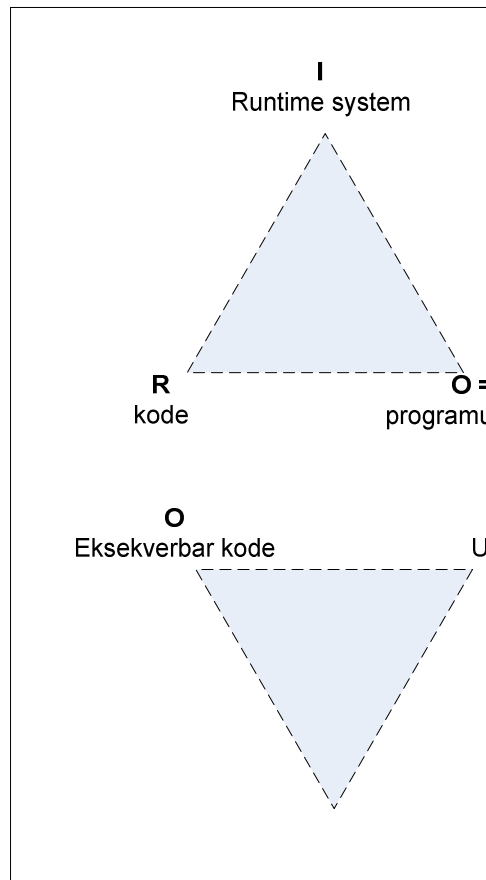
[Foråret]

- Anvendelsesområdet (interpretanten)
 - Opgaver, genstande og personer der bruger systemet
- Problemområdet (objektet)
 - Opgaver, genstande og personer som systemet repræsenterer

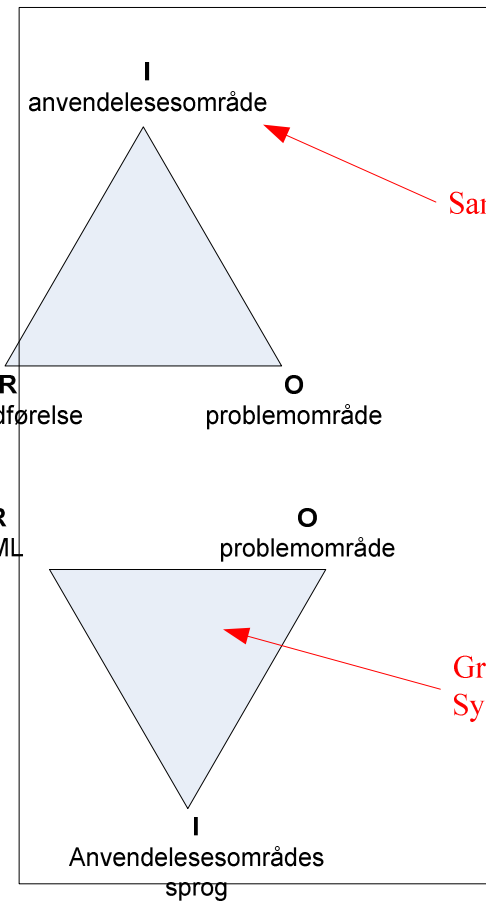


[Foråret]

Intro til programmering



Systemudvikling



Samarbejde med brugere

Grænseflade design
Systembeskrivelse

[Foråret]

- To områder
 - Problemområdet: hvad skal systemet repræsentere?
 - Anvendelsesområdet: hvordan skal det bruges
- **Det er nu forbudt at beskrive implementeringen!**
- Systembeskrivelse:
 - Introduktion til én metode med fokus på design af IT-systemer:
 - Mathiassen, Munk-Madsen, Nielsen og Stage: Objektorienteret analyse & design, Marko, 2001
- Introduktion til bredere metoder med relevans for design af IT-systemer
 - Aktivitetsteori
 - ANT
 - Semiotik
- Grænseflade design: formidling og betjening
 - Traditionelle grænseflader
 - Nye grænseflader, pervasive computing
- Samarbejde med brugere og designvidenskab:
 - Metaforer og mock-ups
 - Prototyper