DD1331

Grundläggande programmering för F1

Några bilder till föreläsning 9

Innehåll

Innehåll

- Klasser

Innehåll

- Klasser
  - Mall för objekt

Innehåll

- Klasser
  - Mall för objekt
  - Egen ny typ

Innehåll

- Klasser
  - Mall för objekt
  - Egen ny typ
  - Inuti klassen

Innehåll

- Klasser
  - Mall för objekt
  - Egen ny typ
  - Inuti klassen
    - Konstruktor
    - Attribut
    - Metoder

Innehåll

- Klasser
  - Mall för objekt
  - Egen ny typ
  - Inuti klassen
    - Konstruktor
    - Attribut
    - Metoder
- Abstrakta datatyper – ADT

Innehåll

- Klasser
  - Mall för objekt
  - Egen ny typ
  - Inuti klassen
    - Konstruktor
    - Attribut
    - Metoder
- Abstrakta datatyper – ADT
  - Viktigt i alla programmeringsparadigmer, särskilt OO

Innehåll

- Klasser
  - Mall för objekt
  - Egen ny typ
  - Inuti klassen
    - Konstruktor
    - Attribut
    - Metoder
- Abstrakta datatyper – ADT
  - Viktigt i alla programmeringsparadigmer, särskilt OO
    - Vad är ADT?

Innehåll

- Klasser
  - Mall för objekt
  - Egen ny typ
  - Inuti klassen
    - Konstruktor
    - Attribut
    - Metoder
- Abstrakta datatyper – ADT
  - Viktigt i alla programmeringsparadigmer, särskilt OO
    - Vad är ADT?
    - Varför är ADT bra?

Innehåll

- Klasser
  - Mall för objekt
  - Egen ny typ
  - Inuti klassen
    - Konstruktor
    - Attribut
    - Metoder
- Abstrakta datatyper – ADT
  - Viktigt i alla programmeringsparadigmer, särskilt OO
    - Vad är ADT?
    - Varför är ADT bra?
  - ADT utifrån

Innehåll

- Klasser
  - Mall för objekt
  - Egen ny typ
  - Inuti klassen
    - Konstruktor
    - Attribut
    - Metoder
- Abstrakta datatyper – ADT
  - Viktigt i alla programmeringsparadigmer, särskilt OO
    - Vad är ADT?
    - Varför är ADT bra?
  - ADT utifrån
  - ADT inifrån

OBJEKT

OBJEKT

Objekt kan ha både data och funktioner

OBJEKT

Objekt kan ha både data och funktioner

- Data som tillhör objektet kallas *attribut*

OBJEKT

Objekt kan ha både data och funktioner

- Data som tillhör objektet kallas *attribut*
- Funktioner som tillhör objektet kallas *metoder*

OBJEKT

Objekt kan ha både data och funktioner

- Data som tillhör objektet kallas *attribut*
- Funktioner som tillhör objektet kallas *metoder*
- Mall för hur objekt ska se ut kallas *klass*

Exempel på Pythons inblyggda klasser

Exempel på Pythons inblyggda klasser

Klass	Attribut	Metodanrop
str `s = "python"`	Alla tecken i strängen	t.ex. `s = s.upper()`
list `fili = [8,0,3,1,2,5]`	Alla element i listan	t.ex. `fili.sort()`

Exempel på Pythons inblyggda klasser

Klass	Attribut	Metodanrop
str `s = "python"`	Alla tecken i strängen	t.ex. `s = s.upper()`
list `fili = [8,0,3,1,2,5]`	Alla element i listan	t.ex. `fili.sort()`

str är *immutable*, kan ej ändras

Exempel på Pythons inblyggda klasser

Klass	Attribut	Metodanrop
str `s = "python"`	Alla tecken i strängen	t.ex. `s = s.upper()`
list `fili = [8,0,3,1,2,5]`	Alla element i listan	t.ex. `fili.sort()`

str är *immutable*, kan ej ändras (`s.upper()` ger nytt str-objekt)

Exempel på Pythons inblyggda klasser

Klass	Attribut	Metodanrop
str `s = "python"`	Alla tecken i strängen	t.ex. `s = s.upper()`
list `fili = [8,0,3,1,2,5]`	Alla element i listan	t.ex. `fili.sort()`

str är *immutable*, kan ej ändras (`s.upper()` ger nytt str-objekt)

list är *mutable*, kan ändras,

Exempel på Pythons inblyggda klasser

Klass	Attribut	Metodanrop
str `s = "python"`	Alla tecken i strängen	t.ex. `s = s.upper()`
list `fili = [8,0,3,1,2,5]`	Alla element i listan	t.ex. `fili.sort()`

str är *immutable*, kan ej ändras (`s.upper()` ger nytt str-objekt)

list är *mutable*, kan ändras, (`fili.sort()` ändrar objektet fili)

Egen klass, exempel 1

Egen klass, exempel 1

Positioner i xy-planet

Egen klass, exempel 1

Positioner i xy-planet

Klass	Attribut	Metoder
Position	x	`show()`
	y	`move()`
		`normalize()`
		`odistance()`

Egen klass, exempel 1

Positioner i xy-planet

Klass	Attribut	Metoder
Position	x	`show()`
	y	`move()`
		`normalize()`
		`odistance()`

Varje Position har egna data, x och y.

Egen klass, exempel 1

Positioner i xy-planet

Klass	Attribut	Metoder
Position	x	`show()`
	y	`move()`
		`normalize()`
		`odistance()`

Varje Position har egna data, x och y.

Varje Position kan utföra `show`, `move`, ...

Klass

(generellt, inte bara Python)

Klass
(generellt, inte bara Python)

- Mall för objekt

Klass

(generellt, inte bara Python)

- Mall för objekt
- Definierar attribut och metoder

Klass

(generellt, inte bara Python)

- Mall för objekt
- Definierar attribut och metoder
  - attribut beskriver objektets tillstånd

Klass

(generellt, inte bara Python)

- Mall för objekt
- Definierar attribut och metoder
  - attribut beskriver objektets tillstånd
  - metoder kan ändra tillståndet

Klass
(generellt, inte bara Python)

- Mall för objekt
- Definierar attribut och metoder
  - attribut beskriver objektets tillstånd
  - metoder kan ändra tillståndet
    men behöver inte göra det

Klass
(generellt, inte bara Python)

- Mall för objekt
- Definierar attribut och metoder
  - attribut beskriver objektets tillstånd
  - metoder kan ändra tillståndet
    men behöver inte göra det
- En klass är en typ

Klass
(generellt, inte bara Python)

- Mall för objekt
- Definierar attribut och metoder
  - attribut beskriver objektets tillstånd
  - metoder kan ändra tillståndet
    men behöver inte göra det
- En klass är en typ
- Många objekt kan skapas från en klass

Klass
(generellt, inte bara Python)

- Mall för objekt
- Definierar attribut och metoder
  - attribut beskriver objektets tillstånd
  - metoder kan ändra tillståndet
    men behöver inte göra det
- En klass är en typ
- Många objekt kan skapas från en klass
- Ett objekt är en instans av en klass

Klassen Position, version 1

Klassen Position, version 1

```
class Position:
    def __init__(self, xin = 0, yin = 0):
        self.x = xin
        self.y = yin

    def show(self):
        print("x =",self.x,"y =",self.y)

    def move(self, dx, dy):
        self.x += dx
        self.y += dy
```

Testkod för Position

Testkod för Position

```
p1 = Position(); p1.show()
p1.move(-3, 8); p1.show()
print()
p2 = Position(42, 236); p2.show()
p2.move(258, 164); p2.show()
```

Testkod för Position

```
p1 = Position(); p1.show()
p1.move(-3, 8); p1.show()
print()
p2 = Position(42, 236); p2.show()
p2.move(258, 164); p2.show()
```

Utskrift

Testkod för Position

```
p1 = Position(); p1.show()
p1.move(-3, 8); p1.show()
print()
p2 = Position(42, 236); p2.show()
p2.move(258, 164); p2.show()
```

Utskrift

```
x = 0 y = 0
x = -3 y = 8

x = 42 y = 236
x = 300 y = 400
```

Klassen Position – längre version

Klassen Position – längre version

```
class Position:
    def __init__(self, xin = 0, yin = 0):
        self.x = xin; self.y = yin

    def move(self, dx, dy):
        self.x += dx; self.y += dy

    def __str__(self):
        return '('+str(self.x)+','+str(self.y)+')'
```

Klassen Position – längre version

```
class Position:
    def __init__(self, xin = 0, yin = 0):
        self.x = xin; self.y = yin

    def move(self, dx, dy):
        self.x += dx; self.y += dy

    def __str__(self):
        return '('+str(self.x)+','+str(self.y)+')'

    def odistance(self):
        xo = self.x; yo = self.y
        return (xo*xo + yo*yo)**0.5
```

Klassen Position – längre version

```
class Position:
    def __init__(self, xin = 0, yin = 0):
        self.x = xin; self.y = yin

    def move(self, dx, dy):
        self.x += dx; self.y += dy

    def __str__(self):
        return '('+str(self.x)+','+str(self.y)+')'

    def odistance(self):
        xo = self.x; yo = self.y
        return (xo*xo + yo*yo)**0.5

    def normalize(self):
        d = self.odistance()
        self.x = self.x/d
        self.y = self.y/d
```

Metoden `__init__`

Metoden `__init__`

- **Konstruktor**

Metoden `__init__`

- **Konstruktor**
- `start = Position(1,2)`
  anropar konstruktorn och skapar ett objekt

Metoden `__init__`

- **Konstruktor**
- `start = Position(1,2)`
  anropar konstruktorn och skapar ett objekt
- Konstruktorn `__init__` anropas automatiskt

Metoden `__init__`

- **Konstruktor**
- `start = Position(1,2)`
  anropar konstruktorn och skapar ett objekt
- Konstruktorn `__init__` anropas automatiskt
- Attributen ges värden

Metoden `__init__`

- **Konstruktor**
- `start = Position(1,2)`
  anropar konstruktorn och skapar ett objekt
- Konstruktorn `__init__` anropas automatiskt
- Attributen ges värden
- En referens till objektet returneras

Metoden `__init__`

- **Konstruktor**
- `start = Position(1,2)`
  anropar konstruktorn och skapar ett objekt
- Konstruktorn `__init__` anropas automatiskt
- Attributen ges värden
- En referens till objektet returneras

Konstruktorn sätter objektets starttillstånd

Metoden `__init__`

- **Konstruktor**
- `start = Position(1,2)`
  anropar konstruktorn och skapar ett objekt
- Konstruktorn `__init__` anropas automatiskt
- Attributen ges värden
- En referens till objektet returneras

Konstruktorn sätter objektets starttillstånd

Konstruktor ska finnas i alla klasser ni skriver i Grupdat

self

`self`

- `self` finns **inuti** varje klass

`self`

- `self` finns **inuti** varje klass
- `self` är en referens till objektet självt

`self`

- `self` finns **inuti** varje klass
- `self` är en referens till objektet självt
- `self`
  ska stå på många ställen **inuti** klassdefinitionen:

`self`

- `self` finns **inuti** varje klass
- `self` är en referens till objektet självt
- `self`
  ska stå på många ställen **inuti** klassdefinitionen:
  - först i parameterlistor: `def metod(self, ...):`

`self`

- `self` finns **inuti** varje klass
- `self` är en referens till objektet självt
- `self`
  ska stå på många ställen **inuti** klassdefinitionen:
  - först i parameterlistor: `def metod(self, ...):`
  - framför användning av attributen: `self.a`

`self`

- `self` finns **inuti** varje klass
- `self` är en referens till objektet självt
- `self`
  ska stå på många ställen **inuti** klassdefinitionen:
  - först i parameterlistor: `def metod(self, ...):`
  - framför användning av attributen: `self.a`
  - framför anrop av metoderna: `self.m()`

`self`

- `self` finns **inuti** varje klass
- `self` är en referens till objektet självt
- `self`
  ska stå på många ställen **inuti** klassdefinitionen:
  - först i parameterlistor: `def metod(self, ...):`
  - framför användning av attributen: `self.a`
  - framför anrop av metoderna: `self.m()`
- `self` används **aldrig utanför** klassen

`self`

- `self` finns **inuti** varje klass
- `self` är en referens till objektet självt
- `self`
  ska stå på många ställen **inuti** klassdefinitionen:
  - först i parameterlistor: `def metod(self, ...):`
  - framför användning av attributen: `self.a`
  - framför anrop av metoderna: `self.m()`
- `self` används **aldrig utanför** klassen

`self` är en formell parameter, annat namn än `self` kan väljas!

Metoden `__str__`

Metoden `__str__`

- Kan definieras i varje klass

Metoden `__str__`

- Kan definieras i varje klass
- Ska **returnera**
  textinformation om objektet

Metoden `__str__`

- Kan definieras i varje klass
- Ska **returnera** textinformation om objektet
- Anropas automatiskt om man skriver ut objektet med `print`

Metoden `__str__`

- Kan definieras i varje klass
- Ska **returnera** textinformation om objektet
- Anropas automatiskt om man skriver ut objektet med `print`

```
origo = Position()
print(origo)
```

```
(0,0)
```

Metoden `__str__`

- Kan definieras i varje klass
- Ska **returnera** textinformation om objektet
- Anropas automatiskt om man skriver ut objektet med `print`

```
origo = Position()
print(origo)
```

```
(0,0)
```

skrivs om `__str__` är rätt definierad

Klassen Position – längre version IGEN, samma som förut

Klassen Position – längre version IGEN, samma som förut

```
class Position:
    def __init__(self, xin = 0, yin = 0):
        self.x = xin; self.y = yin

    def move(self, dx, dy):
        self.x += dx; self.y += dy

    def __str__(self):
        return '('+str(self.x)+','+str(self.y)+')'
```

Klassen Position – längre version IGEN, samma som förut

```
class Position:
    def __init__(self, xin = 0, yin = 0):
        self.x = xin; self.y = yin

    def move(self, dx, dy):
        self.x += dx; self.y += dy

    def __str__(self):
        return '('+str(self.x)+','+str(self.y)+')'

    def odistance(self):
        xo = self.x; yo = self.y
        return (xo*xo + yo*yo)**0.5
```

Klassen Position – längre version IGEN, samma som förut

```
class Position:
    def __init__(self, xin = 0, yin = 0):
        self.x = xin; self.y = yin

    def move(self, dx, dy):
        self.x += dx; self.y += dy

    def __str__(self):
        return '('+str(self.x)+','+str(self.y)+')'

    def odistance(self):
        xo = self.x; yo = self.y
        return (xo*xo + yo*yo)**0.5

    def normalize(self):
        d = self.odistance()
        self.x = self.x/d
        self.y = self.y/d
```

Metoder med namn som `__xxx__`

Metoder med namn som `__xxx__`

- Har speciell betydelse i Python

Metoder med namn som `__xxx__`

- Har speciell betydelse i Python
- Är fördefinierade

Metoder med namn som `__xxx__`

- Har speciell betydelse i Python
- Är fördefinierade
- Kan definieras om av programmeraren

Metoder med namn som `__xxx__`

- Har speciell betydelse i Python
- Är fördefinierade
- Kan definieras om av programmeraren
- Anropas inte direkt

Metoder med namn som `__xxx__`

- Har speciell betydelse i Python
- Är fördefinierade
- Kan definieras om av programmeraren
- Anropas inte direkt
  (utom möjligen för test och demonstration)

Metoder med namn som `__xxx__`

- Har speciell betydelse i Python
- Är fördefinierade
- Kan definieras om av programmeraren
- Anropas inte direkt
  (utom möjligen för test och demonstration)
- Det finns MÅNGA

Metoder med namn som `__xxx__`

- Har speciell betydelse i Python
- Är fördefinierade
- Kan definieras om av programmeraren
- Anropas inte direkt
  (utom möjligen för test och demonstration)
- Det finns MÅNGA men vi tar bara upp

Metoder med namn som `__xxx__`

- Har speciell betydelse i Python
- Är fördefinierade
- Kan definieras om av programmeraren
- Anropas inte direkt
  (utom möjligen för test och demonstration)
- Det finns MÅNGA men vi tar bara upp
  `__init__` och `__str__`

Egen klass, exempel 2

Husdjur

Egen klass, exempel 2

Husdjur

Klass	Attribut		Metoder
Husdjur	`namn`	str	`leka()`
	`glad`	int	`mata()`
	`hungrig`	int	`banna()`
			`__str__()`

Egen klass, exempel 2

Husdjur

Klass	Attribut		Metoder
Husdjur	`namn`	str	`leka()`
	`glad`	int	`mata()`
	`hungrig`	int	`banna()`
			`__str__()`

`namn` har typen `str`

Egen klass, exempel 2

Husdjur

Klass	Attribut		Metoder
Husdjur	`namn`	str	`leka()`
	`glad`	int	`mata()`
	`hungrig`	int	`banna()`
			`__str__()`

`namn` har typen `str`

`glad` har typen `int`,

Egen klass, exempel 2

Husdjur

Klass	Attribut		Metoder
Husdjur	`namn`	str	`leka()`
	`glad`	int	`mata()`
	`hungrig`	int	`banna()`
			`__str__()`

`namn` har typen `str`

`glad` har typen `int`, $< 0 \rightarrow$ ledsen, $\geq 3 \rightarrow$ glad

Egen klass, exempel 2

Husdjur

Klass	Attribut		Metoder
Husdjur	`namn`	str	`leka()`
	`glad`	int	`mata()`
	`hungrig`	int	`banna()`
			`__str__()`

`namn` har typen `str`

`glad` har typen `int`, $< 0 \rightarrow$ ledsen, $\geq 3 \rightarrow$ glad

`hungrig` har typen `int`, $\geq 3 \rightarrow$ hungrig, annars mätt

Egen klass, exempel 2

Husdjur

Klass	Attribut		Metoder
Husdjur	`namn`	str	`leka()`
	`glad`	int	`mata()`
	`hungrig`	int	`banna()`
			`__str__()`

`namn` har typen `str`

`glad` har typen `int`, $< 0 \rightarrow$ ledsen, $\geq 3 \rightarrow$ glad

`hungrig` har typen `int`, $\geq 3 \rightarrow$ hungrig, annars mätt

`leka()`, `mata()`, `banna()` ändrar objektets tillstånd

Klassen Husdjur, ”skelett”

(endast huvuden för metoderna, metodkropparna markeras med ...)

Klassen Husdjur, ”skelett”

(endast huvuden för metoderna, metodkropparna markeras med ...)

```
class Husdjur():
    def __init__(self, namn):
        ...

    def banna(self):
        ...

    def mata(self, antalKakor):
        ...

    def leka(self):
        ...

    def __str__(self):
        ...
```

Klassen Husdjur, fullständig

Klassen Husdjur, fullständig

```
class Husdjur():
    def __init__(self, namn):
        self.namn = namn
        self.hungrig = 2
        self.glad = 3

    def banna(self):
        print("Fy pa dig " + self.namn + "!")
        self.glad -=1

    def mata(self, antalKakor):
        self.hungrig -= antalKakor

    def lek(self):
        self.glad +=1
        self.hungrig +=1
```

fortsätter på nästa sida

Klassen Husdjur, forts.

Klassen Husdjur, forts.

```
    def __str__(self):
        beskrivning = "Jag heter " + self.namn

        if self.hungrig >= 3:
            beskrivning += " och ar hungrig"
        else:
            beskrivning += " och ar matt"

        if self.glad >= 3:
            beskrivning += " och glad"
        elif self.glad < 0:
            beskrivning += " och ledsen"

        return beskrivning
```

Klassen demonstreras på föreläsningen.

ADT – Abstrakt datatyp

Abstraktion – dölj irrelevanta detaljer

ADT – Abstrakt datatyp

Abstraktion – dölj irrelevanta detaljer

- Definition

ADT – Abstrakt datatyp

Abstraktion – dölj irrelevanta detaljer

- Definition
  - Data och operationer med dolt lagringsformat

ADT – Abstrakt datatyp

Abstraktion – dölj irrelevanta detaljer

- Definition
  - Data och operationer med dolt lagringsformat
- Exempel:

ADT – Abstrakt datatyp

Abstraktion – dölj irrelevanta detaljer

- Definition
  - Data och operationer med dolt lagringsformat
- Exempel:
  - Pythons inbyggda typer str och list

ADT – Abstrakt datatyp

Abstraktion – dölj irrelevanta detaljer

- Definition
  - Data och operationer med dolt lagringsformat
- Exempel:
  - Pythons inbyggda typer str och list
  - Alla klasser som har attribut och metoder

ADT – Abstrakt datatyp

Abstraktion – dölj irrelevanta detaljer

- Definition
  - Data och operationer med dolt lagringsformat
- Exempel:
  - Pythons inbyggda typer str och list
  - Alla klasser som har attribut och metoder
- Metoderna i ADT:n kallas för dess gränssnitt eller interface

Hur och Varför ADT ?

Hur och Varför ADT ?

(antag en **konstruktör** ocn en **användare** av ADT:n)

Hur och Varför ADT ?

(antag en **konstruktör** ocn en **användare** av ADT:n)

- **Konstruktören** och **användaren** kommer överens om gränssnittet och jobbar sedan med var sitt problem.

Hur och Varför ADT ?

(antag en **konstruktör** ocn en **användare** av ADT:n)

- **Konstruktören** och **användaren** kommer överens om gränssnittet och jobbar sedan med var sitt problem.
- **Användaren** behöver inte förstå innehållet i ADT:n, bara hur man använder den.

Hur och Varför ADT ?

(antag en **konstruktör** ocn en **användare** av ADT:n)

- **Konstruktören** och **användaren** kommer överens om gränssnittet och jobbar sedan med var sitt problem.
- **Användaren** behöver inte förstå innehållet i ADT:n, bara hur man använder den.
- **Användaren** kan inte missbruka detaljinformation i ADT:n utan måste hålla sig till gränssnittet.

Hur och Varför ADT ?

(antag en **konstruktör** ocn en **användare** av ADT:n)

- **Konstruktören** och **användaren** kommer överens om gränssnittet och jobbar sedan med var sitt problem.
- **Användaren** behöver inte förstå innehållet i ADT:n, bara hur man använder den.
- **Användaren** kan inte missbruka detaljinformation i ADT:n utan måste hålla sig till gränssnittet.
- **Konstruktören** kan förbättra ADT:n utan att **användarens** kod behöver ändras, bara gränssnittet bibehålls.

Hur och Varför ADT ?

(antag en **konstruktör** ocn en **användare** av ADT:n)

- **Konstruktören** och **användaren** kommer överens om gränssnittet och jobbar sedan med var sitt problem.
- **Användaren** behöver inte förstå innehållet i ADT:n, bara hur man använder den.
- **Användaren** kan inte missbruka detaljinformation i ADT:n utan måste hålla sig till gränssnittet.
- **Konstruktören** kan förbättra ADT:n utan att **användarens** kod behöver ändras, bara gränssnittet bibehålls.
- Lättare att överblicka program om delproblem är lösta separat.

Hur och Varför ADT ?

(antag en **konstruktör** ocn en **användare** av ADT:n)

- **Konstruktören** och **användaren** kommer överens om gränssnittet och jobbar sedan med var sitt problem.
- **Användaren** behöver inte förstå innehållet i ADT:n, bara hur man använder den.
- **Användaren** kan inte missbruka detaljinformation i ADT:n utan måste hålla sig till gränssnittet.
- **Konstruktören** kan förbättra ADT:n utan att **användarens** kod behöver ändras, bara gränssnittet bibehålls.
- Lättare att överblicka program om delproblem är lösta separat.

Fortsättning följer ...

Hur och varför ADT ?

(fortsättning)

Hur och varför ADT ?

(fortsättning)

- En ADT kan ofta användas i flera tillämpningar. Sådana återanvändbara klasser sparas i bibliotek. **Exempel?**

Hur och varför ADT ?

(fortsättning)

- En ADT kan ofta användas i flera tillämpningar.
  Sådana återanvändbara klasser sparas i bibliotek.
  Exempel?
- Nödvändigt att strukturera med ADT när många personer tillsammans utvecklar stora program.

Hur och varför ADT ?

(fortsättning)

- En ADT kan ofta användas i flera tillämpningar.
  Sådana återanvändbara klasser sparas i bibliotek.
  Exempel?
- Nödvändigt att strukturera med ADT när många personer tillsammans utvecklar stora program.
- **Konstruktören** och **användaren** kan vara samma person, vid olika tidpunkter !!!