Lær Python dag 3 - modul 1

Filer

Jonas Bamse Andersen

Institut for Matematik og Datalogi - IMADA
Syddansk Universitet

Indhold

1. Recap

2. Dictionaries

3. Filer

Recap

Hvad lærte I sidst?

Diskutér to minutter med sidemanden, hvad I lærte sidst.

- Hvad er en løkke?
- Hvad er en liste?
- Hvad kan vi med strenge?
- Andet?

Objekter og metoder

I har prøvet at bruge funktioner som:

```
x = [1, 2]
x.append(3)
print(x)
```

Objekter og metoder

I har prøvet at bruge funktioner som:

```
x = [1, 2]
x.append(3)
print(x)
```

Output:

```
[1, 2, 3]
```

Objekter og metoder

I har prøvet at bruge funktioner som:

```
x = [1, 2]
x.append(3)
print(x)
```

Output:

```
[1, 2, 3]
```

Har I undret jer over hvad punktummet betyder?

Objekter og metoder

I har prøvet at bruge funktioner som:

```
x = [1, 2]
x.append(3)
print(x)
```

Output:

```
[1, 2, 3]
```

Har I undret jer over hvad punktummet betyder? Hvorfor skriver man ikke:

```
append(x, 3)
```

Objekter og metoder

Hvad sker der hvis man skriver:

```
x = 2
x.append(3)
```

Objekter og metoder

Hvad sker der hvis man skriver:

```
x = 2
x.append(3)
```

Output:

```
...

AttributeError: 'int' object has no attribute 'append'
```

Objekter og metoder

Hvad sker der hvis man skriver:

```
x = 2
x.append(3)
```

Output:

```
...

AttributeError: 'int' object has no attribute 'append'
```

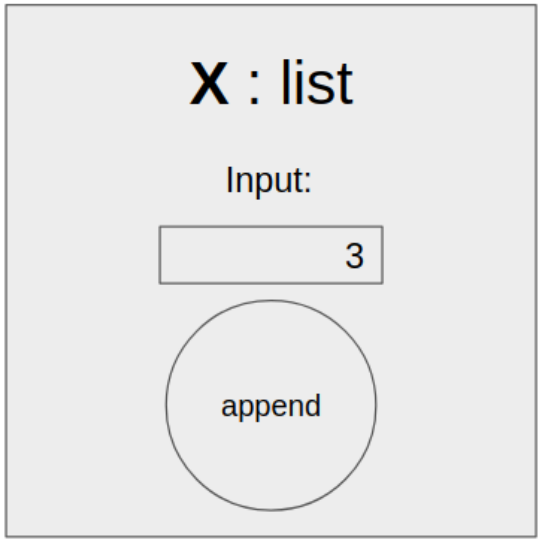
Men så må det være tilfældet at - `'list' object has attribute 'append'`.

Hvad end det så betyder.

Objekter og metoder

Man kan tænke på det som et "interface" på "kassen".

```
x = [1,2]
x.append(3)
```

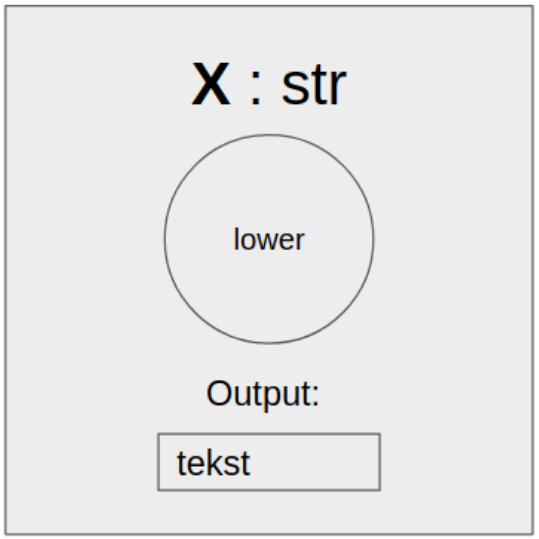


Eksempler på flere metoder kan findes på: `https://docs.python.org/3/tutorial/datastructures.html`

Objekter og metoder

Et andet eksempel kunne være

```
x = "TEKST"
x.lower()
```



Eksempler på flere metoder kan findes på:

`https://docs.python.org/3/library/stdtypes.html#str`

Dictionaries

Dictionaries - Gem data og find det igen

Navnet dictionary kommer tydeligvis fra det engelske ord for ordbog, og grunden til det er at man bruger et ord til at slå op med (key) for at finde en bestemt værdi (value).

Dictionaries - Gem data og find det igen

Navnet dictionary kommer tydeligvis fra det engelske ord for ordbog, og grunden til det er at man bruger et ord til at slå op med (key) for at finde en bestemt værdi (value).

I de lister vi har set på bruges et bestemt tal (indeks) til at finde en bestemt værdi (value), dvs. værdierne har en bestemt rækkefølge.

Dictionaries - Gem data og find det igen

Navnet dictionary kommer tydeligvis fra det engelske ord for ordbog, og grunden til det er at man bruger et ord til at slå op med (key) for at finde en bestemt værdi (value).

I de lister vi har set på bruges et bestemt tal (indeks) til at finde en bestemt værdi (value), dvs. værdierne har en bestemt rækkefølge.

Dictionaries minder meget om, men vi ved ikke noget om rækkefølgen på samme måde, tilgengæld slipper vi for at holde styr på, hvilken placering en værdi har.

Dictionaries - Gem data og find det igen

Navnet dictionary kommer tydeligvis fra det engelske ord for ordbog, og grunden til det er at man bruger et ord til at slå op med (key) for at finde en bestemt værdi (value).

I de lister vi har set på bruges et bestemt tal (indeks) til at finde en bestemt værdi (value), dvs. værdierne har en bestemt rækkefølge.

Dictionaries minder meget om, men vi ved ikke noget om rækkefølgen på samme måde, tilgengæld slipper vi for at holde styr på, hvilken placering en værdi har.

I Python er "{}"og ":"vigtige for at lave et dictionary.

Her er et eksempel på en dansk–engelsk ordbog over nogle dyr.

```
translate = {"hund" : "dog", "kat" : "cat", "hest" : "horse"}
```

Dictionaries - Gem data og find det igen

Nu kan vi bruge vores dictionary til at slå oversættelserne op. Hvis man har et par "x": "y", så bruger man den forreste streng, "x", som index for at finde "y".

```
translate = {"hund" : "dog",
             "kat"  : "cat",
             "hest" : "horse"}
print( translate ["hest"])
```

Dictionaries - Gem data og find det igen

Nu kan vi bruge vores dictionary til at slå oversættelserne op. Hvis man har et par "x": "y", så bruger man den forreste streng, "x", som index for at finde "y".

```
translate = {"hund" : "dog",
             "kat"  : "cat",
             "hest" : "horse"}
print( translate ["hest" ])
```

```
horse
```

Dictionaries - Gem data og find det igen

Vi kan nemt tilføje nye dyr til vores ordbog.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
translate ["ged"] = "goat"
translate ["gris"] = "pig"
print( translate ["ged"])
```

Dictionaries - Gem data og find det igen

Vi kan nemt tilføje nye dyr til vores ordbog.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
translate ["ged"] = "goat"
translate ["gris"] = "pig"
print( translate ["ged"])
```

```
goat
```

Dictionaries - Gem data og find det igen

Hvad sker der hvis vi vil oversætte noget vi ikke har i vores ordbog?

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
print( translate ["hamster"])
```

Dictionaries - Gem data og find det igen

Hvad sker der hvis vi vil oversætte noget vi ikke har i vores ordbog?

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
print( translate ["hamster"])
```

```
KeyError

    Traceback (most
    recent call last )
<ipython-input
    -20-55577e0fe5a3>
    in <module>()
-----> 1 print(translate
    ["hamster"])

KeyError: 'hamster'
```

Dictionaries - Gem data og find det igen

Heldigvis kan vi tjekke om en værdi findes i vores dictionary.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
def safeTranslate(animal):
    if (animal in translate):
        print( translate [animal])
    else:
        print("<" + animal +
              "> not found")

safeTranslate("hamster")
safeTranslate("hund")
```

Dictionaries - Gem data og find det igen

Heldigvis kan vi tjekke om en værdi findes i vores dictionary.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
def safeTranslate(animal):
    if (animal in translate):
        print( translate [animal])
    else:
        print("<" + animal +
              "> not found")

safeTranslate("hamster")
safeTranslate("hund")
```

```
<hamster> not found
dog
```

Dictionaries - Gem data og find det igen

Hvis vi skal løbe igennem alle ting i listen kan vi gøre det på flere forskellige måder.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
for navn in translate :
  print("dansk: " +
        navn +
        ", engelsk: " +
        translate [navn])
```

Dictionaries - Gem data og find det igen

Hvis vi skal løbe igennem alle ting i listen kan vi gøre det på flere forskellige måder.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
for navn in translate :
  print("dansk: " +
        navn +
        ", engelsk: " +
        translate [navn])
```

```
dansk: hund, engelsk: dog
dansk: kat, engelsk: cat
dansk: hest, engelsk: horse
```

Dictionaries - Gem data og find det igen

I stedet for at skulle "slå op"i vores ordbog i hvert gennemløb af loopet, kan vi få key - value parret direkte ved at bruge `.items()` på dictionariet.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
print( translate .items())
```

Dictionaries - Gem data og find det igen

I stedet for at skulle "slå op"i vores ordbog i hvert gennemløb af loopet, kan vi få key - value parret direkte ved at bruge `.items()` på dictionariet.

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
print( translate .items())
```

```
dict_items ([
  ('hund', 'dog'),
  ('kat', 'cat'),
  ('hest', 'horse')])
```

Dictionaries - Gem data og find det igen

Vi kan også gøre det i et loop, hvor vi får værdierne direkte ud:

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
for key, val in translate.items():
  print("dansk: " +
        key +
        ", engelsk: " +
        val)
```

Dictionaries - Gem data og find det igen

Vi kan også gøre det i et loop, hvor vi får værdierne direkte ud:

```
translate = {"hund" : "dog",
             "kat" : "cat",
             "hest" : "horse"}
for key, val in translate.items():
  print("dansk: " +
        key +
        ", engelsk: " +
        val)
```

```
dansk: hund, engelsk: dog
dansk: kat, engelsk: cat
dansk: hest, engelsk: horse
```

Dictionaries - Gem data og find det igen

Dictionaries kan bruges til meget andet, og tingene (keys) behøver ikke være strenge, det kan være hvad som helst. Her er et eksempel på en repræsentation af en person.

```
person = {"name" : "Bamse",
          "age": 24,
          "height": 187}

if (person["height"] >= 182):
    print("You are tall "
          + person["name"])
else:
    print("You can still grow "
          + person["name"])
```

Dictionaries - Gem data og find det igen

Dictionaries kan bruges til meget andet, og tingene (keys) behøver ikke være strenge, det kan være hvad som helst. Her er et eksempel på en repræsentation af en person.

```
person = {"name" : "Bamse",
          "age": 24,
          "height": 187}

if (person["height"] >= 182):
    print("You are tall "
          + person["name"])
else:
    print("You can still grow "
          + person["name"])
```

```
You are tall
    Bamse
```

Dictionaries - Gem data og find det igen

Et andet eksempel kan være et dictionary som holder styr på varers priser i en butik.

```
price = {"Banana" : 10,
         "Milk" : 5,
         "Bread" : 20,
         "Salt" : 2}
print(price["Banana"])
```

Dictionaries - Gem data og find det igen

Et andet eksempel kan være et dictionary som holder styr på varers priser i en butik.

```
price = {"Banana" : 10,
         "Milk" : 5,
         "Bread" : 20,
         "Salt" : 2}
print(price["Banana"])
```

```
10
```

Dictionaries - Gem data og find det igen

Hvis vi tidligere ville have lavet et program der skulle tælle bogstaver i en lang streng, hvordan ville vi så have gjort?

Dictionaries - Gem data og find det igen

Hvis vi tidligere ville have lavet et program der skulle tælle bogstaver i en lang streng, hvordan ville vi så have gjort?

Her er et forslag:

```
s = "Lorem ipsum dolor sit amet..."
counts = [0] * 26 # [0,0,0,0,...]
for c in s.lower():
    if c == "a":
        counts[0] = counts[0] + 1
    elif c == "b":
        counts[1] = counts[1] + 1
    elif ...
.
.
.
```

Dictionaries - Gem data og find det igen

Man kunne også gøre mange andre smartere ting end det, men det smarteste er nok at bruge dictionaries.

Dictionaries - Gem data og find det igen

Man kunne også gøre mange andre smartere ting end det, men det smarteste er nok at bruge dictionaries.

```
s = "fem flade flødeboller på et fladt flødebollefad"
d = {}
for c in s:
    if c not in d:
        d[c] = 0
    d[c] = d[c] + 1
print(d)
```

Strenge og Dictionaries - Sidste bemærkninger

Som altid har jeg kun vist jer et lillebitte udsnit af hvilke funktioner der findes. Så husk, Google er kun få klik væk.

Filer

Filer og programmering

Filer bruges grundlæggende til to ting:

- Input til program (f.eks. til databehandling)
- Gemme output fra program (resultater, logs, . . .)

Åben/luk en fil

Vi åbner en fil med metoden `open()` som returnerer et filobjekt:

```
f = open(<file>, <mode>)
```

Åben/luk en fil

Vi åbner en fil med metoden `open()` som returnerer et filobjekt:

f = **open**(<**file**>, <mode>)

`open` tager parametrene `file` og `mode`:

- File: Stien til den fil som skal åbnes (string).
- Mode: Hvordan skal filen bruges
  - "r" for at læse (read)
  - "w" for at skrive (write)
  - "r+" for begge dele
  - "a" for at lægge til (append)

Åben/luk en fil

Vi åbner en fil med metoden `open()` som returnerer et filobjekt:

```
f = open(<file>, <mode>)
```

`open` tager parametrene `file` og `mode`:

- File: Stien til den fil som skal åbnes (string).
- Mode: Hvordan skal filen bruges
  - "r" for at læse (read)
  - "w" for at skrive (write)
  - "r+" for begge dele
  - "a" for at lægge til (append)

Når man er færdig med filen er det god stil at lukke den:

```
f.close()
```

Skriv til en fil

Skriv til filen `hello.txt` på følgende vis:

```
f = open("hello.txt", "w")
f.write("Hello World!")
f.close()
```

Hvis vi skriver til en fil som ikke eksisterer oprettes en ny.

Skriv til en fil

Vi skal selv indsætte vores linjeskift.

```
f = open("hello.txt", "w")
f.write("Lær python!!!\n")
f.write("IMADA SDU")
f.close()
```

Hvordan mon vores fil ser ud nu?

Skriv til en fil

Vi skal selv indsætte vores linjeskift.

```
f = open("hello.txt", "w")
f.write("Lær python!!!\n")
f.write("IMADA SDU")
f.close()
```

Hvordan mon vores fil ser ud nu?

"w" overskriver filen uden at stille spørgsmål.

"a" bruges til at tilføje til en fil.

Læs fra en fil

Vi kan læse filen `hello.txt` på følgende vis:

```
f = open("hello.txt", "r")
print(f.read())
f.close()
```

Læs fra en fil

Vi kan også læse en fil linje for linje, her `README.md`:

```
f = open("sample_data/README.md", "r")
for line in f:
    print(line)
f.close()
```

Kan også gøres med `while` og `readline`.

```
f = open("sample_data/README.md", "r")
line = f.readline()
while line:
    print(line)
    line = f.readline()
f.close()
```

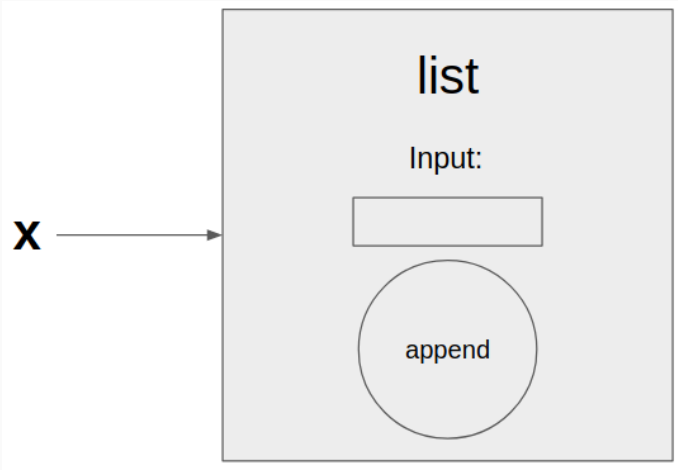
Læs fra en fil

`rstrip()` fjerner whitespace for enden af linjen, så undgår vi de ekstra linjeskift.

```
f = open("sample_data/README.md", "r")
for line in f:
    print(line.rstrip())
f.close()
```

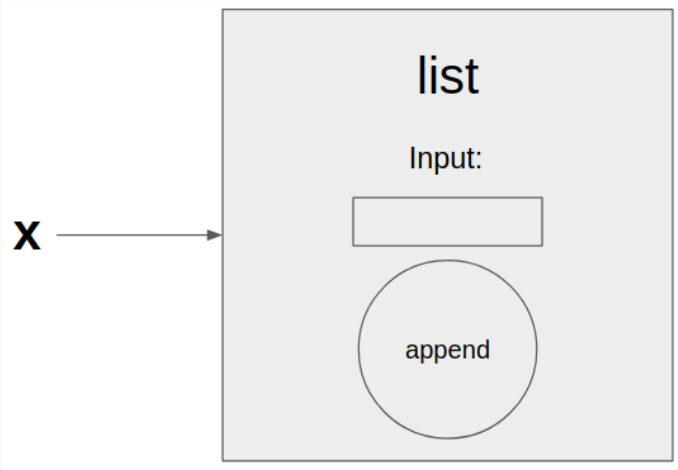
Objekter og metoder

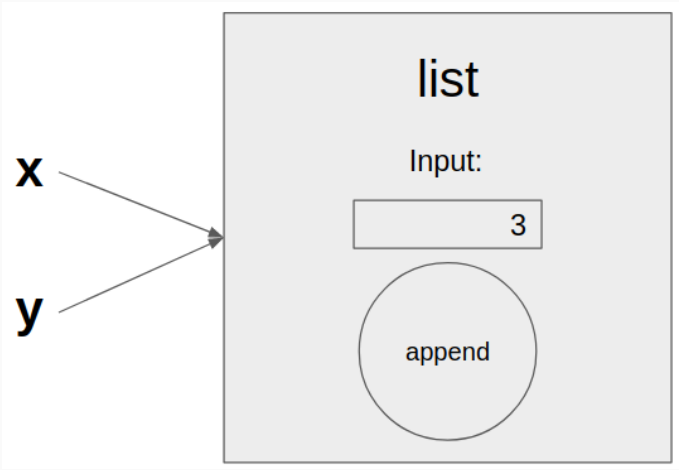
Nu kan det være brugbart at tænke på et variabelnavn i denne sammenhæng som en reference til et objekt, og man kan have flere referencer til det samme objekt.



Objekter og metoder

Nu kan det være brugbart at tænke på et variabelnavn i denne sammenhæng som en reference til et objekt, og man kan have flere referencer til det samme objekt.





Objekter og metoder

Det var bl.a. tilfældet med lister:

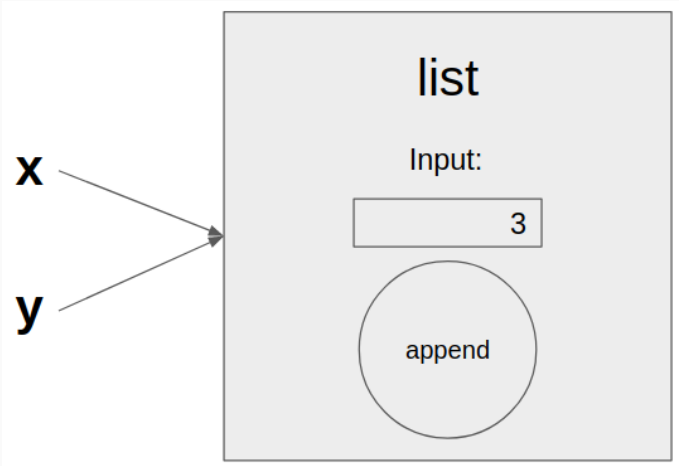
```
x = [1,2]
y = x
print(y)
x.append(3)
print(y)
```

Objekter og metoder

Det var bl.a. tilfældet med lister:

```
x = [1,2]
y = x
print(y)
x.append(3)
print(y)
```

```
[1, 2]
[1, 2, 3]
```

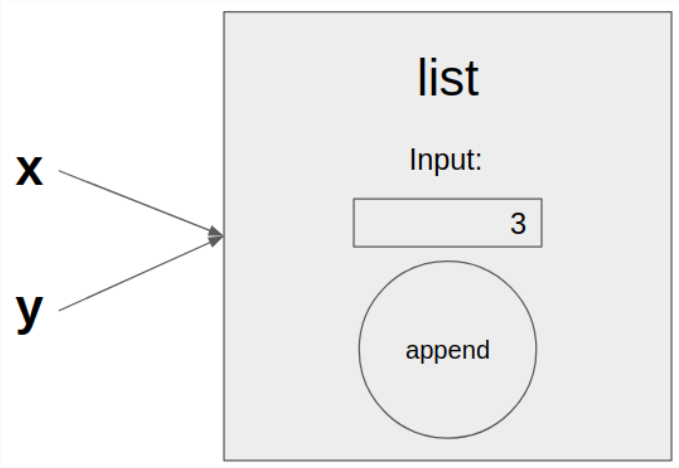


Objekter og metoder

Det var bl.a. tilfældet med lister:

```
x = [1,2]
y = x
print(y)
x.append(3)
print(y)
```

```
[1, 2]
[1, 2, 3]
```



Hvis det er ved at smelte din hjerne lidt, så husk at det er helt normalt.

Objekter og metoder

Det samme gælder filer.

```
f1 = open("sample_data/README.md", "r")
f2 = f1
print(f1.readline().rstrip())
print(f1.readline().rstrip())
print(f2.readline().rstrip())
f1.close()
```

Exceptions

Hvad sker der hvis vi prøver at læse en fil som ikke eksisterer?

```
f = open("fail.txt")
```

Exceptions

Hvad sker der hvis vi prøver at læse en fil som ikke eksisterer?

```
f = open("fail.txt")
```

```
...
FileNotFoundError: [Errno 2] No such file or directory: 'fail.
    txt'
```

Programmet stopper med en fejl!

Exceptions

Vi kan fange denne fejl (exception), og give en bedre fejlmeddelelse, eller alt efter fejlen prøve at rette op på den.

Dette gøres via `try` og `except`:

```
try:
    f = open("fail.txt", "r")
    print(f.read())
    f.close()
except:
    print("Noget gik galt.")
```

Exceptions

Vi kan også fange en specifik fejl:

```
try:
    f = open("fail.txt", "r")
    print(f.read())
    f.close()
except FileNotFoundError:
    print("Filen kunne ikke findes")
```

Exceptions

Vi kan også fange en specifik fejl:

```
try:
    f = open("fail.txt", "r")
    print(f.read())
    f.close()
except FileNotFoundError:
    print("Filen kunne ikke findes")
```

Find den fejltype du ønsker her:

`https://docs.python.org/3/library/exceptions.html`

En anden måde at håndtere filer

Vi kan slippe for at tænke på at lukke filen med et `with` statement.

```
with open("sample_data/README.md", "r") as f:
  for line in f:
    print(line.rstrip())
```

En anden måde at håndtere filer

Vi skal dog stadig fange fejl, hvis de opstår:

```
try:
  with open("fail.txt", "r") as f:
    for line in f:
      print(line.rstrip())
except:
  print("Noget gik galt.")
```

Afsluttende bemærkning

I har nu lært om bl.a.

- Typer & variabler
- If-sætninger
- Funktioner
- Lister
- Løkker
- Strenge
- Dictionaries
- Filer

Hvilket i princippet er mere end rigeligt til at kunne gøre "alt".

Afsluttende bemærkning

Det vigtigste vi ikke har lært om er hvordan man laver sine egne "klasser", sine egne typer.

Afsluttende bemærkning

Det vigtigste vi ikke har lært om er hvordan man laver sine egne "klasser", sine egne typer.

Derudover er der masser af detaljer om de ting vi har lært som også kan udforskes.

Afsluttende bemærkning

Det vigtigste vi ikke har lært om er hvordan man laver sine egne "klasser", sine egne typer.

Derudover er der masser af detaljer om de ting vi har lært som også kan udforskes.

Læs andres kode, læs bøger, se videoer, lav tutorials, lav et projekt og find hjælp på nettet.