Masterclass Datalogi

Programmering og Python

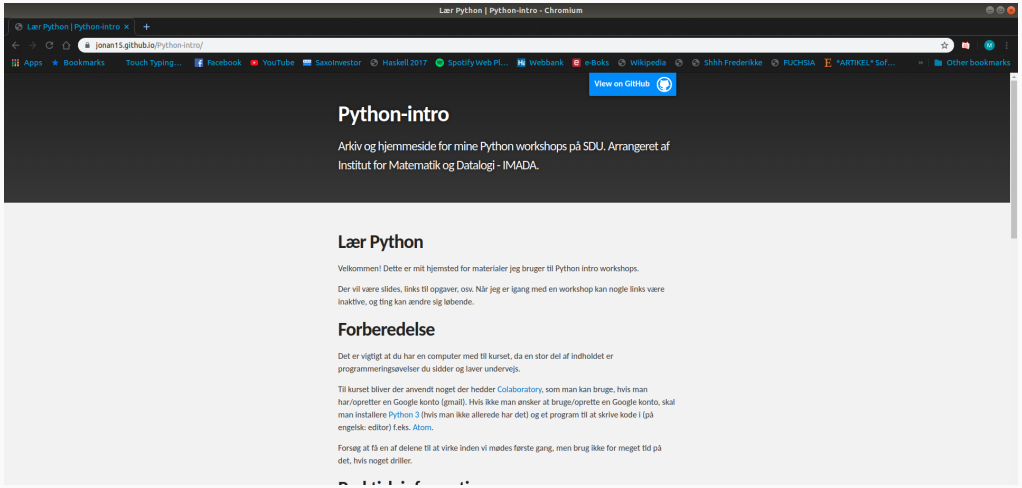
Frederick, Louie & Bamse

Ungdommens Naturvidenskabelige Forening UNF

Hjemmeside

Vi har lavet en lille hjemmeside:

`jonan15.github.io/Masterclass`

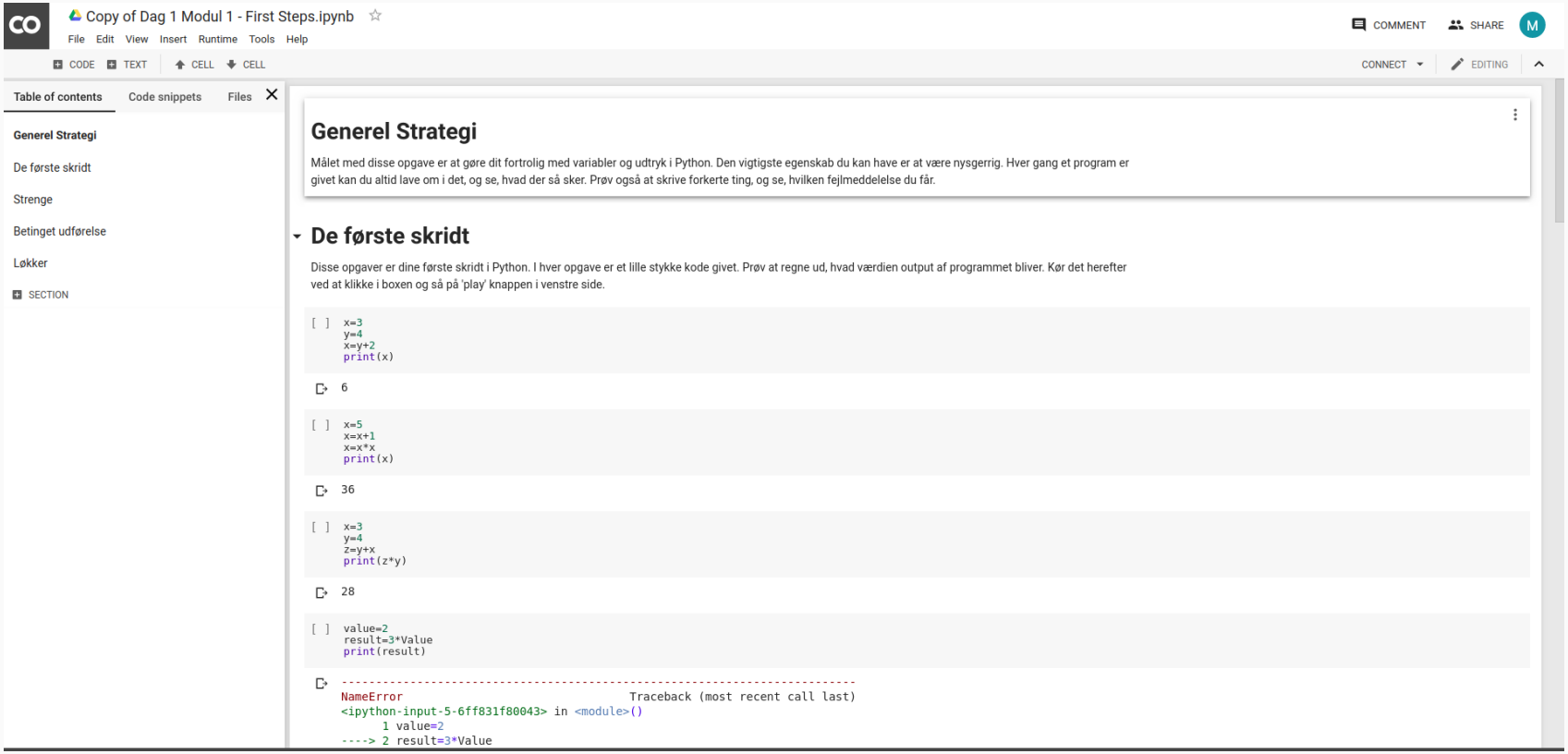


Her findes slides, opgaver og eventuelle tips & tricks.

Colaboratory

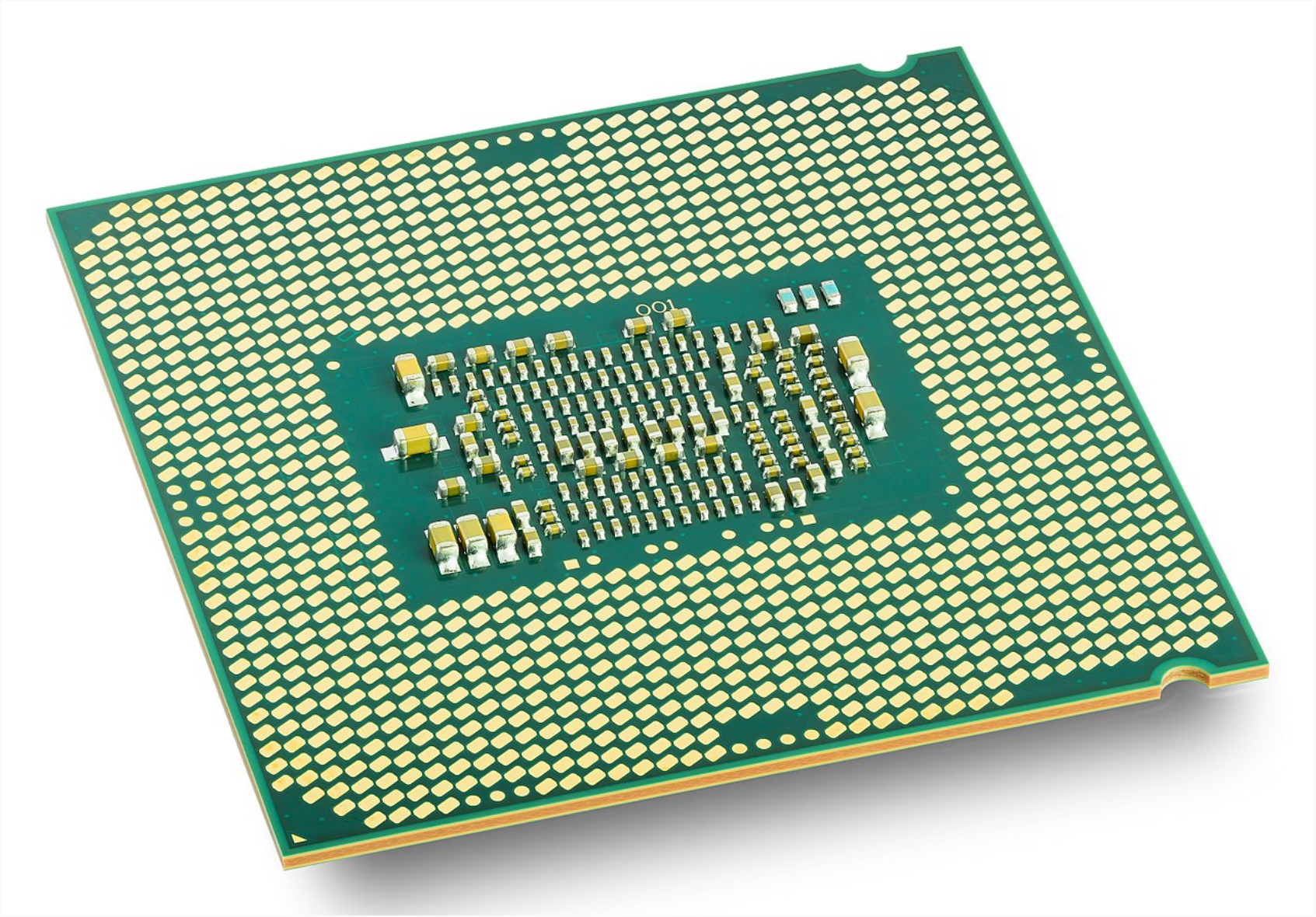
Online editor og interpreter (fortolker).

Kræver en Google Account.



Man kan også installere Python på sin egen computer.

Hvad er Python?



Hvad er Python?

Hvad er en computer?

Hvad er Python?

Hvad er en computer?

Hvad er et program?

Hvad er Python?

Hvad er en computer?

Hvad er et program?

Program = sekvens af instruktioner.

Hvad er Python?

Hvad er en computer?

Hvad er et program?

Program = sekvens af instruktioner.

Computere er dumme! De gør kun som de får besked på...

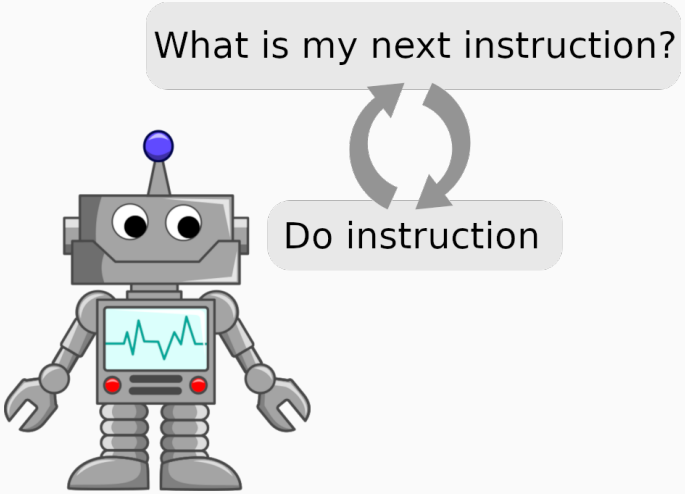
Hvad er Python?

Hvad er en computer?

Hvad er et program?

Program = sekvens af instruktioner.

Computere er dumme! De gør kun som de får besked på...



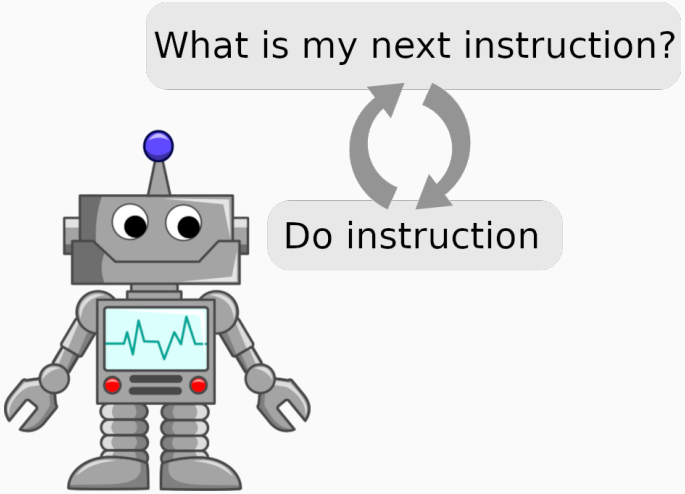
Hvad er Python?

Hvad er en computer?

Hvad er et program?

Program = sekvens af instruktioner.

Computere er dumme! De gør kun som de får besked på...



Tilgengæld er de lynende hurtige.

Hvad er Python?

Python er et programmeringssprog.

Sproget bestemmer hvilke instruktioner man kan skrive.

Men sproget skal oversættes/fortolkes.

Hvad er Python?

Python er et programmeringssprog.

Sproget bestemmer hvilke instruktioner man kan skrive.

Men sproget skal oversættes/fortolkes.

1 linje $\approx$ 1 instruktion.

Programmet oversættes fra toppen og nedad.

Programmer

Når vi snakker om programmer kan vi bl.a. snakke om:

- Instruktioner: Hvad selve programmet består af.

Programmer

Når vi snakker om programmer kan vi bl.a. snakke om:

- Instruktioner: Hvad selve programmet består af.
- Input: "Data"som programmet skal forholde sig til.

Programmer

Når vi snakker om programmer kan vi bl.a. snakke om:

- Instruktioner: Hvad selve programmet består af.
- Input: "Data"som programmet skal forholde sig til.
- Output: Resultatet af programmet, f.eks. tekst/grafik på en skærm eller en ny fil.

Disclaimer

Programmering skal øves - det tager tid.

Vi forsøger at vise jer de første basale ting, men I når ikke at blive mestre, desværre :(

Disclaimer

Programmering skal øves - det tager tid.

Vi forsøger at vise jer de første basale ting, men I når ikke at blive mestre, desværre :(

Men I bliver nok bedre end jeres venner :)

Målet er at begynde at få jer til at tænke som programmører, at I lærer noget nyt og at I har det sjovt.

Programmering i Python

Hello, World!

Det er tid til at se det første program køre.

Typer, variabler og udtryk

Datatyper - et programs enheder

Følgende er de mest brugte primitive typer:

Type	Datatyper	Eksempel
`str`	String (streng)	"Batman"
`int`	Integer (heltal)	42
`float`	Float (kommatal)	3.14
`bool`	Boolean	True

Datatyper - et programs enheder

En type kan tjekkes i python via:

```
type(< type_der_skal_tjekkes >)
```

Program:

```
type(42.0)
```

Output:

```
float
eller
<class ' float '>
```

Operatorer

Følgende beskriver de basale operatorer anvendt på tal:

Operation	Beskrivelse	Eksempel	Resultat
$+$	Læg to operander sammen	$2 + 2$	4
$-$	Træk to operander fra hinanden	$50 - 8$	42
$*$	Gang to operander	$3 * 4$	12
$/$	Division mellem to operander	$10/3$	$3.333\ldots$

Operatorer

Følgende beskriver de basale operatorer anvendt på tal:

Operation	Beskrivelse	Eksempel	Resultat
$+$	Læg to operander sammen	$2 + 2$	4
$-$	Træk to operander fra hinanden	$50 - 8$	42
$*$	Gang to operander	$3 * 4$	12
$/$	Division mellem to operander	$10/3$	$3.333\ldots$

Disse operatorer kan måske anvendes på andet?...

Streng-operatorer

Plus?:

```
print("Bat" + "man")
```

Streng-operatorer

Plus?:

```
print("Bat" + "man")
```

```
Batman
```

Streng-operatorer

Plus?:

```
print("Bat" + "man")
```

```
Batman
```

Minus?:

```
print("Batman" - "man
    ")
```

Streng-operatorer

Plus?:

```
print("Bat" + "man")
```

```
Batman
```

Minus?:

```
print("Batman" - "man
    ")
```

```
Traceback (most recent
    call last):
File "<stdin>", line 1,
    in <module>
TypeError: unsupported
    operand type(s) for
    -: 'str' and 'str'
```

Streng-operatorer

Gange?:

```
print("Ha" * 3)
```

Streng-operatorer

Gange?:

```
print("Ha" * 3)
```

```
HaHaHa
```

Streng-operatorer

Gange?:

```
print("Ha" * 3)
```

```
HaHaHa
```

Dividere?:

```
print("hej" / 3)
```

Streng-operatorer

Gange?:

```
print("Ha" * 3)
```

```
HaHaHa
```

Dividere?:

```
print("hej" / 3)
```

```
Traceback (most recent
    call last):
File "<stdin>", line 1,
    in <module>
TypeError: unsupported
    operand type(s) for
    /: 'str' and 'int'
```

Variabler

En variabel er en "beholder" som kan gemme en værdi.

En variabel har et navn:

Tilladte variabelnavne
x
et_navn
TEST
var2
_hej

Forbudte variabelnavne
1var
en.var
-var

Navnet bruges til at referere til værdien senere.

Det er en god idé at give navne der giver mening!

Variabler

En tildeling gemmer noget i den givne beholder/variabel.
Tildeling til en variabel sker på følgende vis:

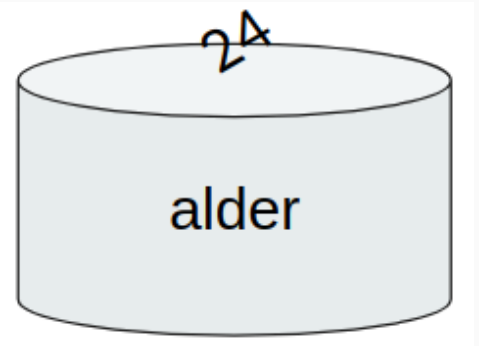
```
<navn> = <værdi>
```

Tildeling af et heltal:

```
alder  = 24
```

Tildeling af en streng:

```
hilsen  = "hej med jer"
```



Variabler

Man refererer til værdien gemt i en variabel ved at skrive variablens navn. Navnet bliver erstattet af værdien den refererer til.

```
yndlingshelt = "Superman"
print( yndlingshelt )
```

```
Superman
```

Variabler

Man refererer til værdien gemt i en variabel ved at skrive variablens navn. Navnet bliver erstattet af værdien den refererer til.

```
yndlingshelt = "Superman"
print( yndlingshelt )
```

```
Superman
```

Bemærk forskellen på at referere og på tekst-strenge

```
yndlingshelt = "Superman"
print(" yndlingshelt ")
```

```
yndlingshelt
```

Variabler

En variabel kan opdateres undervejs, men husker kun det sidste der er blevet lagt i.

```
yndlingshelt = "Superman"
print( yndlingshelt )
yndlingshelt = "Batman"
print( yndlingshelt )
```

```
Superman
Batman
```

Udtryk

Et udtryk er en kombination af værdier, variabler og operatorer

Eksempler på udtryk:

```
5
```

```
x = 3


x + 5
```

```
x = 2
y = 3


(1 + x) * 5 - y
```

Regnereglerne fra tal overholdes.

Print

Udskriv udtryk og variablers værdier med `print`-funktionen:

Program:

```
print("Batman")
print(42)
x = 23
print(x)
```

Output:

```
Batman
42
23
```

Print kan bruges til output fra et program, men også til fejlfinding af ens program (debugging).

Input fra brugeren

Input fra brugeren tages på følgende vis:

```
<variabel> = input(<Beskrivende streng>)
```

Input fra brugeren

Input fra brugeren tages på følgende vis:

```
<variabel> = input(<Beskrivende streng>)
```

Eksempel 1:

```
navn = input("Indtast dit navn?")
```

Input fra brugeren

Input fra brugeren tages på følgende vis:

```
<variabel> = input(<Beskrivende streng>)
```

Eksempel 1:

```
navn = input("Indtast dit navn?")
```

Eksempel 2:

```
alder = input("Alder?")
print("Din alder om to år er")
print(alder+2)
```

Input fra brugeren

Input fra brugeren tages på følgende vis:

```
<variabel> = input(<Beskrivende streng>)
```

Eksempel 1:

```
navn = input("Indtast dit navn?")
```

Eksempel 2:

```
alder = input("Alder?")
print("Din alder om to år er")
print(alder+2)
```

```
...
TypeError: must be
    str, not int
```

Input fra brugeren

Vi bliver nødt til at lave teksten om:

```
alder = input("Alder?")
print("Din alder om to år er")
print(int(alder)+2)
```

Input fra brugeren

Vi bliver nødt til at lave teksten om:

```
alder = input("Alder?")
print("Din alder om to år er")
print(int(alder)+2)
```

```
Alder?25
27
```

Input fra brugeren

Vi bliver nødt til at lave teksten om:

```
alder = input("Alder?")
print("Din alder om to år er")
print(int(alder)+2)
```

```
Alder?25
27
```

Eller:

```
alder = int(input("Alder?"))
print("Din alder om to år er")
print(alder+2)
```

Input fra brugeren

Vi bliver nødt til at lave teksten om:

```
alder = input("Alder?")
print("Din alder om to år er")
print(int(alder)+2)
```

```
Alder?25
27
```

Eller:

```
alder = int(input("Alder?"))
print("Din alder om to år er")
print(alder+2)
```

```
Alder?25
27
```

Kommentarer

Kommentarer i koden er tekst som ignoreres når koden køres.
Disse er kun til ære for den der læser koden.

```
print("Superman") # en kommentar som beskriver denne
    instruktion

# en fritstående kommentar som beskriver den følgende
    kode
print("Batman")
```

Gode kommentarer hjælper når man ser på sin kode lang tid efter man har skrevet den.

Moduler

Mange ting er allerede implementeret i Python af dygtige programmører. Disse funktioner er tilgængelige via moduler (aka. biblioteker).
Vi kan f.eks. benytte `math`-biblioteket til at lave klassiske matematiske operationer:

Program:

```
import math
x = 25
y = math.sqrt(x)
print(y)
```

Output:

```
5.0
```

Se dokumentation for alle matematikfunktioner:
`https://docs.python.org/3/library/math.html`

Conditionals

Boolske udtryk - sandt og falsk

Vi har ofte brug for at sammenligne ting, for at tage beslutninger. For at sammenligne om ting er lig med hinanden bruges dobbelt lig med, ==.

```
print(4 == 4)
print(4 == 2)
print(type(True))
```

Boolske udtryk - sandt og falsk

Vi har ofte brug for at sammenligne ting, for at tage beslutninger. For at sammenligne om ting er lig med hinanden bruges dobbelt lig med, ==.

```
print(4 == 4)
print(4 == 2)
print(type(True))
```

```
True
False
<type 'bool'>
```

Boolske udtryk - sandt og falsk

Her er en liste af sammenligningsoperatorer.

```
x == y  # er x lig med y?
x != y  # er x forskellig fra y?
x < y   # er x mindre end y?
x <= y  # er x mindre end eller lig med y?
x > y
x >= y
```

Eksempler, alle giver True.

```
4 > 2
2+2 == 4
3 != 4
"hej" == "h" + "ej"
```

Boolske udtryk - sandt og falsk

Man kan også bruge `not` til at skifte `True` til `False` og omvendt:

not $(3 < 4)$
True $==$ (**not** False)

Boolske udtryk - sandt og falsk

Man kan også bruge `not` til at skifte `True` til `False` og omvendt:

```
not (3 < 4)
True == (not False)
```

Output

```
False
True
```

If-sætninger - at tage et valg

Meget ofte vil man gerne tage en beslutning, hvor man gør forskellige ting alt efter situationen, f.eks. kan man ikke bruge 10 kr, **hvis** man har ikke har så mange penge i sin pung.

Til dette har programmering if-sætninger, f.eks. i følgende eksempel, **(bemærk indrykningen!)**:

```
money = 52
if (money >= 10):
    money = money - 10
    print("You bought an expensive banana")

print("You have " + str(money) + " money left")
```

If-sætninger - at tage et valg

Man kan også tilføje noget man vil gøre hvis det første ikke var tilfældet:

```
money = 52
if (money >= 10):
    money = money - 10
    print("You bought an expensive banana")
else:
    print("Go earn some more money!")

print("You have " + str(money) + " money left")
```

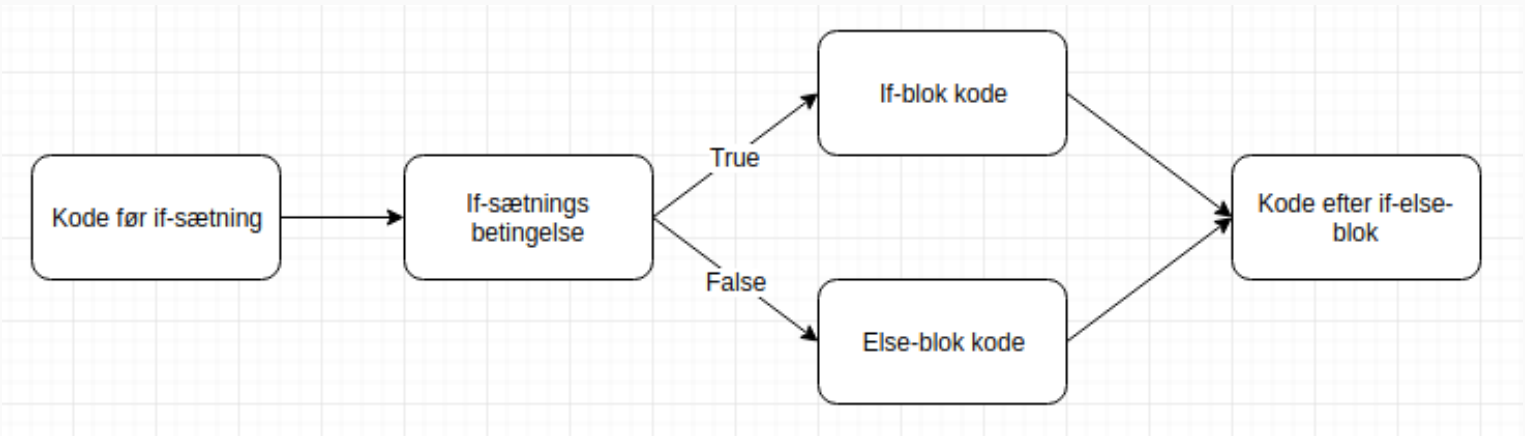
If-sætninger - at tage et valg

En if-sætning er opbygget af en eller flere betingelser og en eller flere "kroppe" af kode.

```
if (<betingelse>):
    #kode
elif (<betingelse):
    #kode
else:
    #kode
```

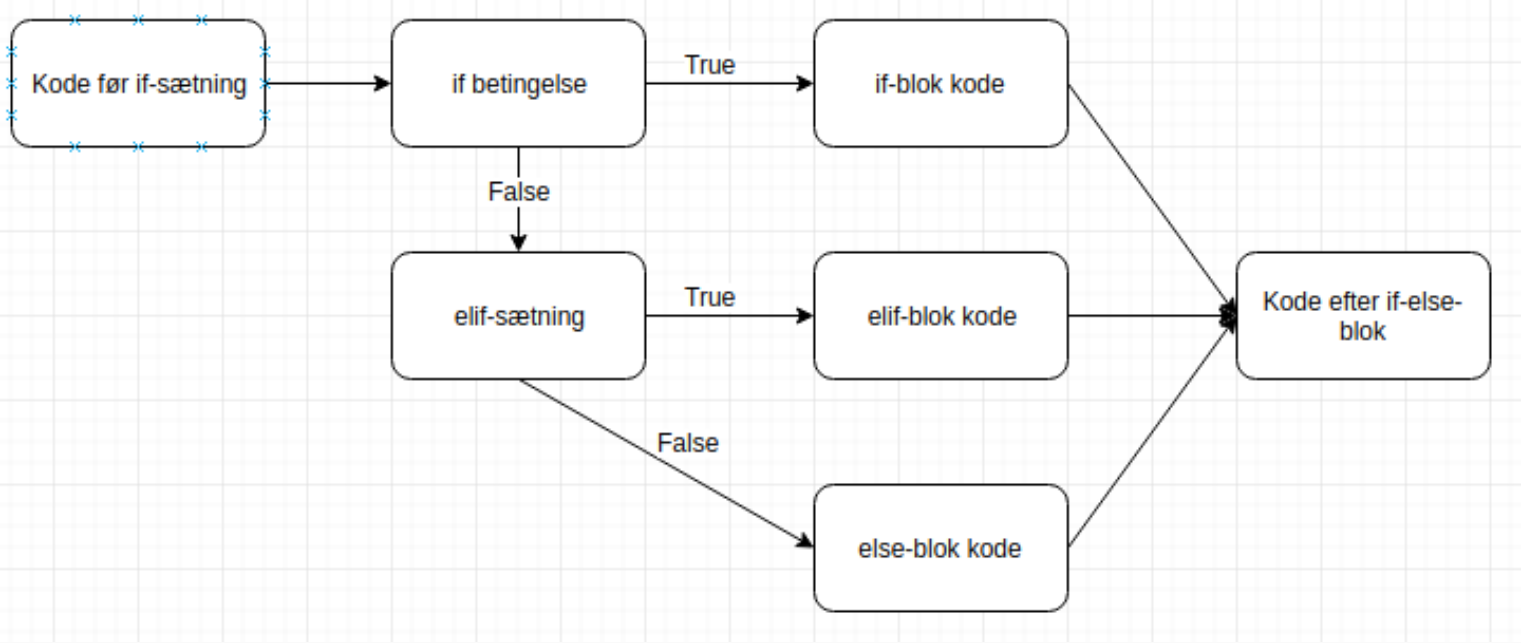
If-sætninger - at tage et valg

Man kan også tænke på det som et flow diagram.



If-sætninger - at tage et valg

Og med `elif` ser det således ud:



Husk at der kan være lige så mange `elif`'s man har lyst, men den første der er sand vil blive valgt og de andre vil så ikke blive tjekket.

If-sætninger - at tage et valg

Måske skal der gælde mere end én ting før man vil gøre noget. F.eks. køber man kun bananen, hvis man både er over 18 og har penge nok.

$$\begin{aligned} \text{age} &= 20 \\ \text{money} &= 42 \end{aligned}$$

If-sætninger - at tage et valg

Måske skal der gælde mere end én ting før man vil gøre noget. F.eks. køber man kun bananen, hvis man både er over 18 og har penge nok. Her er en måde at løse det på:

```
age = 20
money = 42
if (age >= 18):
    if (money >= 10):
        money = money - 10
        print("You bought an expensive banana")
```

If-sætninger - at tage et valg

Der kan også være situationer, hvor bare én af flere ting er nok, for at udføre noget. F.eks. kan enten en fødselsdag eller en bommert være anledning til kage. Hvordan vil man så skrive sit program?

```
birthday = False
blunder = True
```

If-sætninger - at tage et valg

Der kan også være situationer, hvor bare én af flere ting er nok, for at udføre noget. F.eks. kan enten en fødselsdag eller en bommert være anledning til kage. Hvordan vil man så skrive sit program?

If-sætninger - at tage et valg

Der kan også være situationer, hvor bare én af flere ting er nok, for at udføre noget. F.eks. kan enten en fødselsdag eller en bommert være anledning til kage. Hvordan vil man så skrive sit program? Måske sådan her:

```
birthday = False
blunder = True
caketime = False
if (birthday):
  caketime = True
if (blunder):
  caketime = True
if (caketime):
  print("It's cake time!")
```

If-sætninger - at tage et valg

Heldigvis findes der en bedre måde. Man kan sammensætte boolske udtryk med `and` og `or`. Med `and` skal begge sider være opfyldt før det giver True, med `or` skal mindst én side være True før det giver True.

If-sætninger - at tage et valg

Heldigvis findes der en bedre måde. Man kan sammensætte boolske udtryk med `and` og `or`. Med `and` skal begge sider være opfyldt før det giver True, med `or` skal mindst én side være True før det giver True.

```
age = 20
money = 42
if (age >= 18 and money >= 42):
    money = money - 10
    print("You bought an expensive banana")
```

```
birthday = False
blunder = True
if (birthday or blunder):
    print("It's cake time!")
```

Løkker

Tæl til 5

Hvordan kan vi skrive kode som tæller fra 1-5 (printer tallene)?

Tæl til 5

Hvordan kan vi skrive kode som tæller fra 1-5 (printer tallene)?

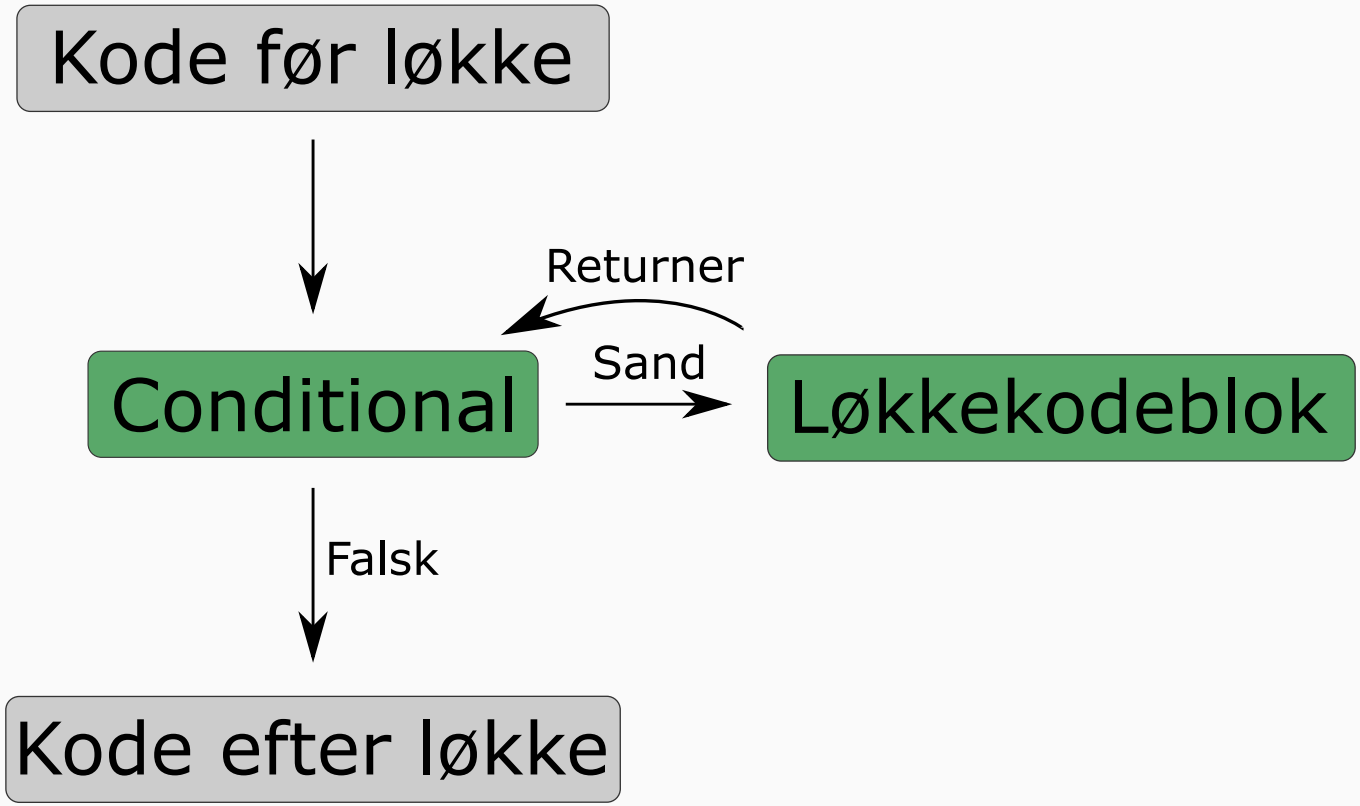
```
print(1)
print(2)
print(3)
print(4)
print(5)
```

```
1
2
3
4
5
```

Virker umiddelbart lidt besværligt, hvad hvis det er en større mængde instruktioner vi vil have gentaget flere gange?
Eller hvad med et variabelt antal gange?

Løkker

Vi kan gøre det med løkker!



For loop

Den type løkke i vil møde her i weekenden er for-løkker.

I Python fungerer det som et for each loop. Altså for hvert element i en eller anden mængde.

```
for <variabel> in <mængde>:
    <instruktion>
    <instruktion>
    <instruktion>
```

For loop

Den type løkke i vil møde her i weekenden er for-løkker.

I Python fungerer det som et for each loop. Altså for hvert element i en eller anden mængde.

```
for <variabel> in <mængde>:
    <instruktion>
    <instruktion>
    <instruktion>
```

Eksempel:

```
for i in range(4):
    print(i)
```

```
0
1
2
3
```

For loop

Men hvad hvis vi gerne vil tælle fra 1 til 5?

For loop

Men hvad hvis vi gerne vil tælle fra 1 til 5?

```
for i in range(1, 6):
    print(i)
```

```
1
2
3
4
5
```

Læg mærke til grænserne. Off-by-one er en klassisk fejl at lave...

Begynd at vænne jer til at tælle fra 0.

For loop

Vi kan også tage skridt af større end en:

```
for i in range(1, 10, 3):
    print(i)
```

```
1
4
7
```

Nested loops

Man kan også have løkker i løkker:

```
for i in range(2):
    for j in range(3):
        print(i, j)
```

Nested loops

Man kan også have løkker i løkker:

```
for i in range(2):
    for j in range(3):
        print(i, j)
```

```
0 0
0 1
0 2
1 0
1 1
1 2
```

Afslutning

Nu har I lært noget om:

- Typer
- Variabler
- Aritmetik og operatorer
- Betinget udførsel
- For-loops
- Kommentarer
- Moduler
- ...

Så nu skal I ikke lytte til os mere.

Det er tid til at lege!