EN UKE I INF100

Tors	Fre

	Man	Tirs	Ons	Tors	Fre
8-10					
10-12					
12-14					
14-16	Fo.les.				

	Man	Tirs
8-10		
10-12		
12-14		
14-16		

Fredag 16:00
Påmelding åpner
for fysisk oppmøte
til forelesning.

Mandag 16:00
Ukesoppgaver publiseres

I løpet av uken
- 2x2 timer gruppe +
- 4-10 timer egenstudie

Mandag 23:59
Innleveringsfrist

LAB1

- sqrt og cmath
- hvorfor vises det None på slutten?
- hardkoding

- du må ha 100% riktig på CodeGrade for å få bestått
- du kan levere så mange ganger du vil (siste innlevering telles)
- ekstra gruppetimer i dag fra 16 – 18

UNIVERSITETET I BERGEN

RETURVERDI

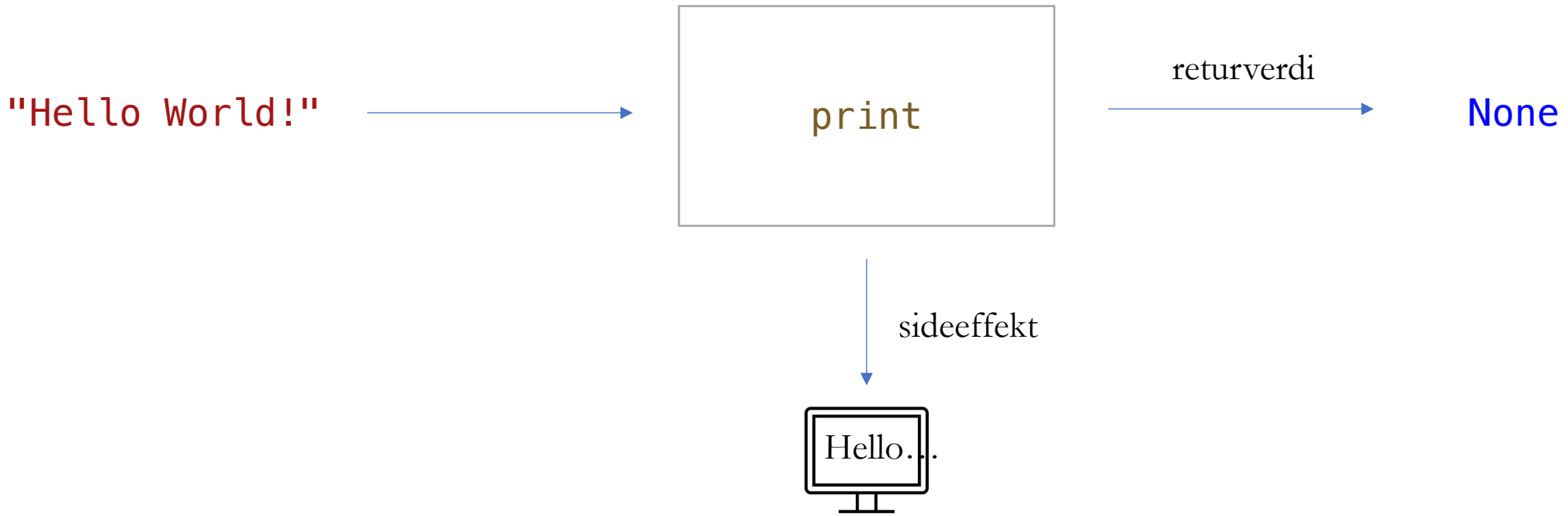
INF100

HØST 2022

Torstein Strømme

SIDEEFFEKT vs. RETURVERDI

```
print("Hello World!")
```



SIDEEFFEKT vs. RETURVERDI

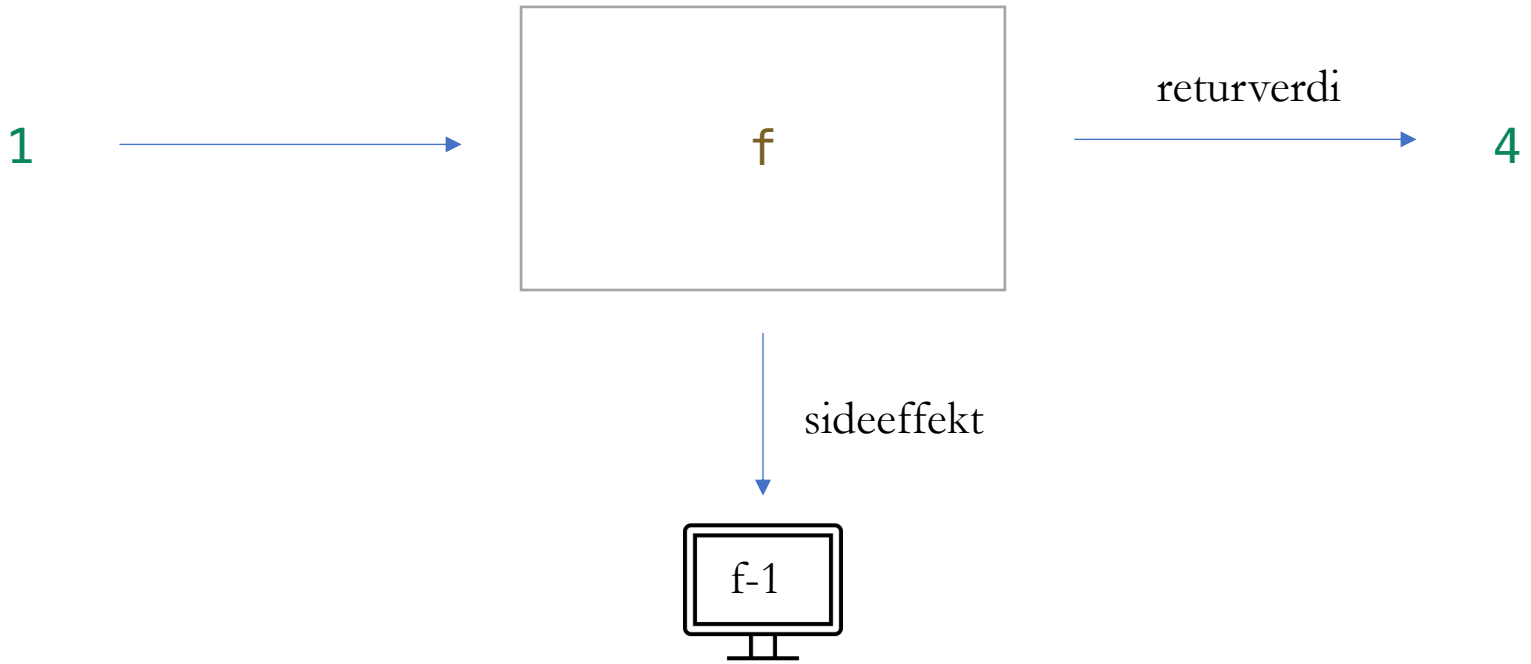
```
max(1, 2, 3)
```

ingen sideeffekt

SIDEEFFEKT vs. RETURVERDI

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x
```

```
f(1)
```



SIDEEFFEKT vs. RETURVERDI

- Alle effekter på datamaskinen sin tilstand utenom returverdi kalles sideeffekter
  - For eksempel, utskrift til skjermen
  - Kan også være å lese input, endre globale variabler, eller mutering av objekter (som vi ikke har lært noe om enda)

- Et funksjonskall evaluerer til sin returverdi

```
x = max(1, 2, 3) # x får verdien 3 (returverdi fra funksjonskallet)
y = print(1, 2, 3) # y får verdien None (returverdi fra funksjonskallet)
z = f(1) # z får verdien 4 (returverdi fra funksjonskallet)
```

KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```

VARIABLER

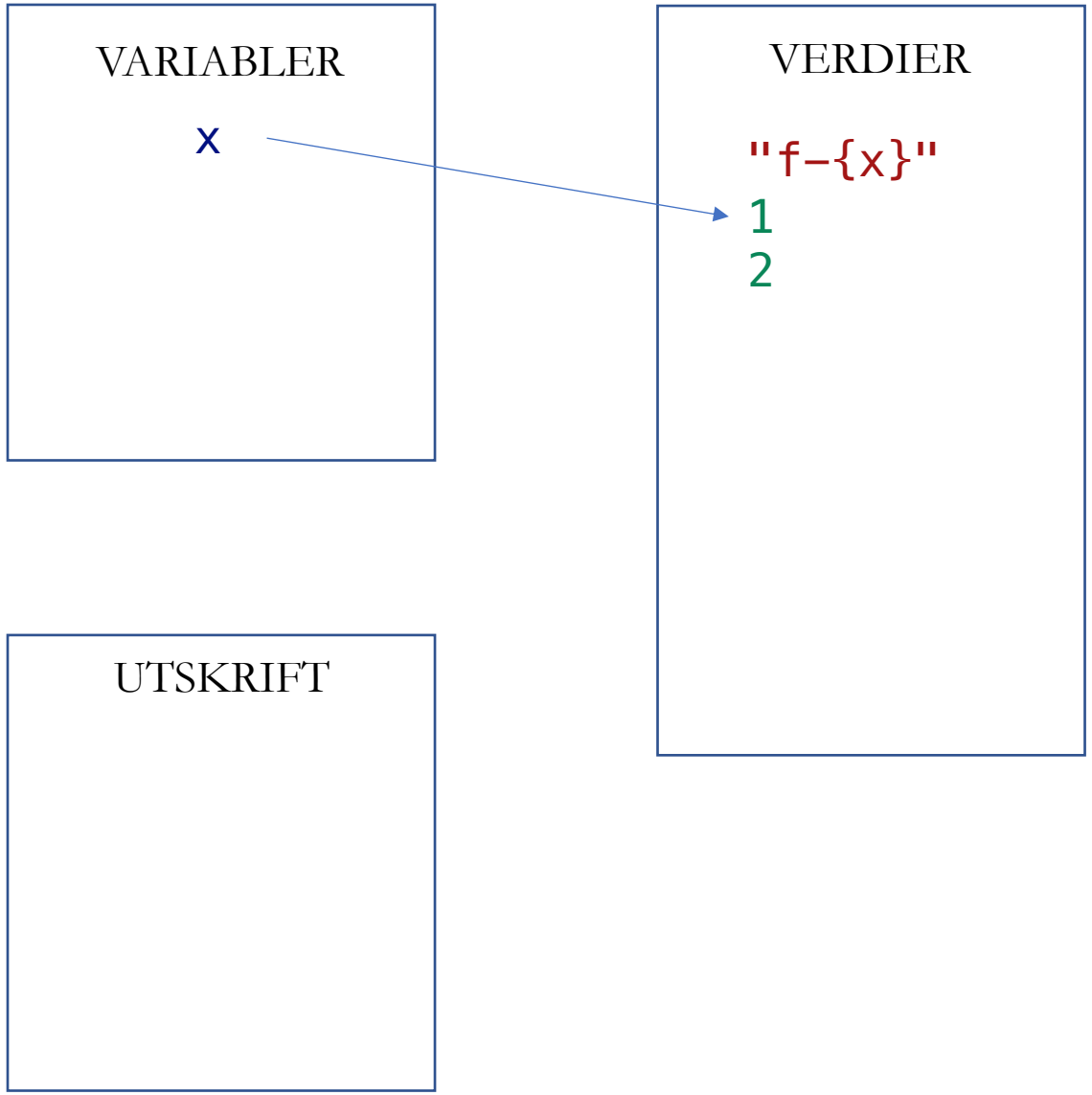
VERDIER

"f-{x}"
1
2

UTSKRIFT

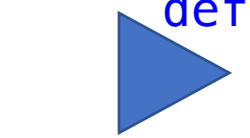
KODESPORING

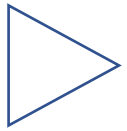
```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```

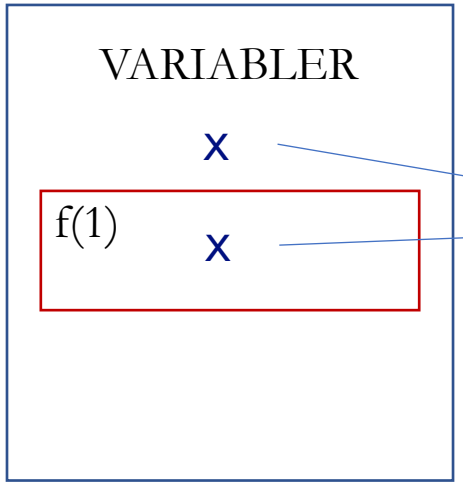


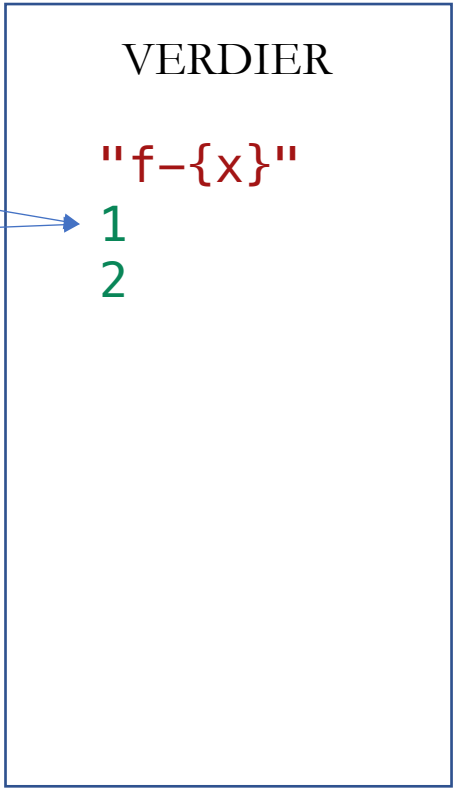
KODESPORING

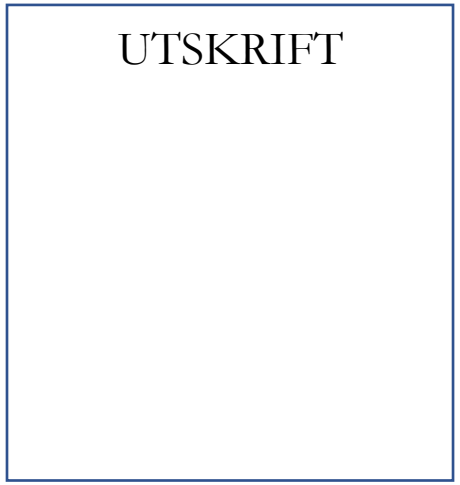
```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```





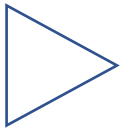


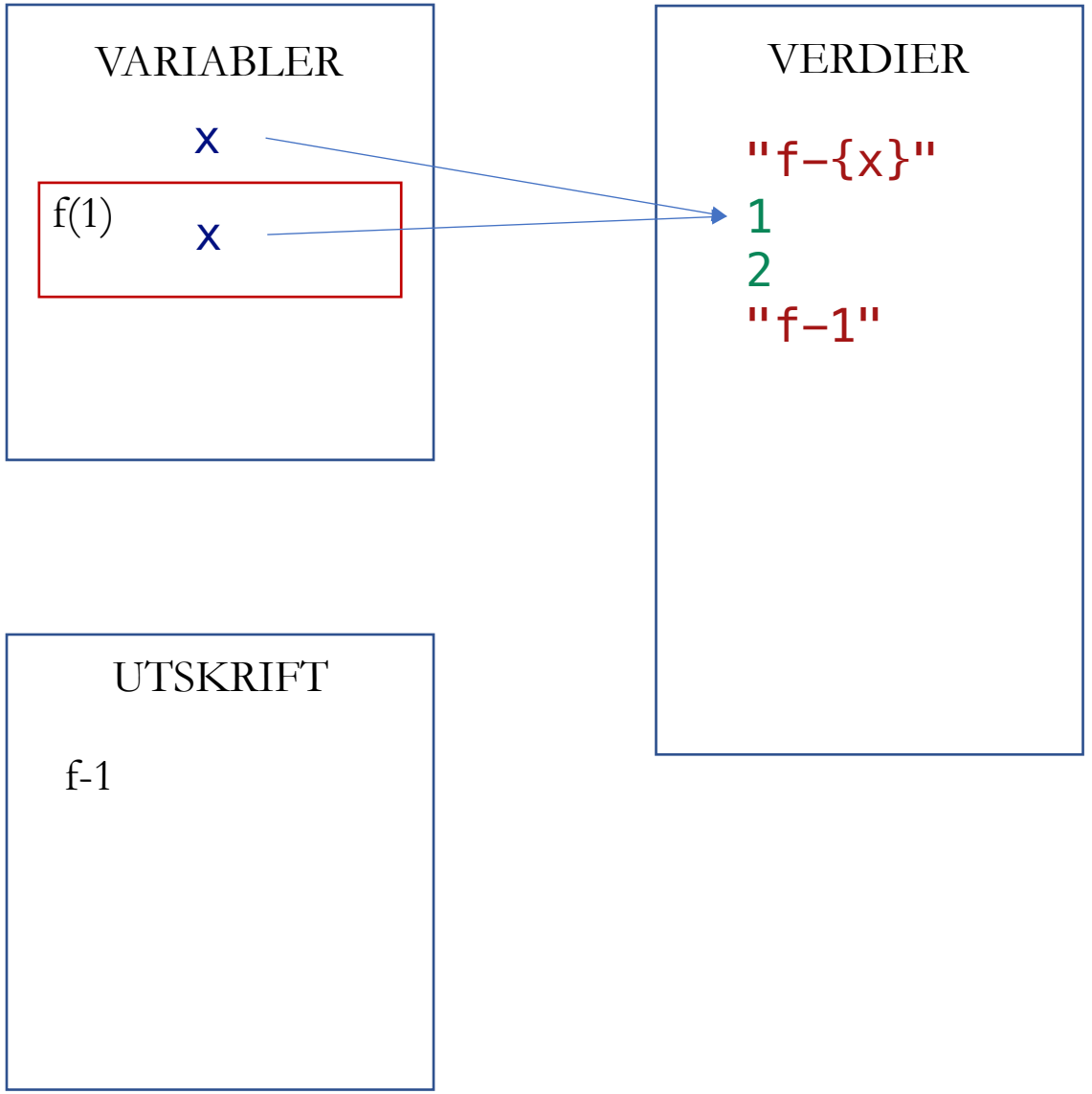




KODESPORING

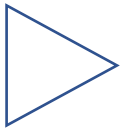
```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```

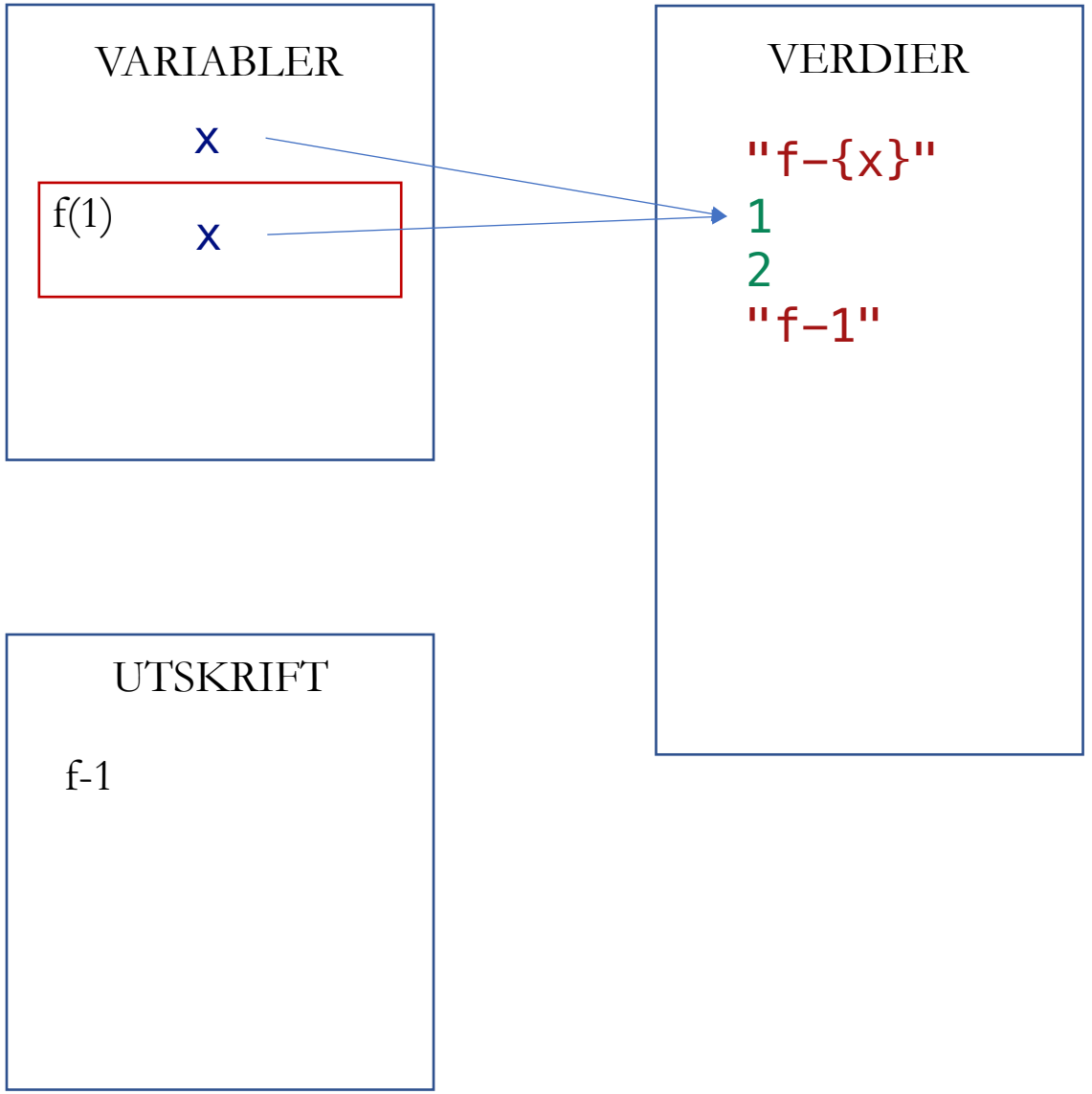




KODESPORING

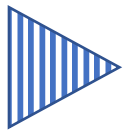
```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```

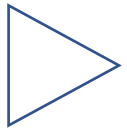


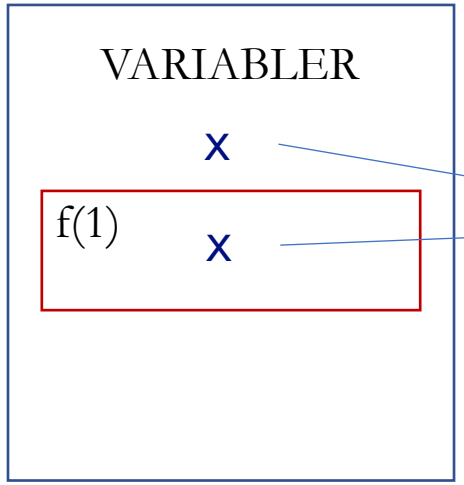


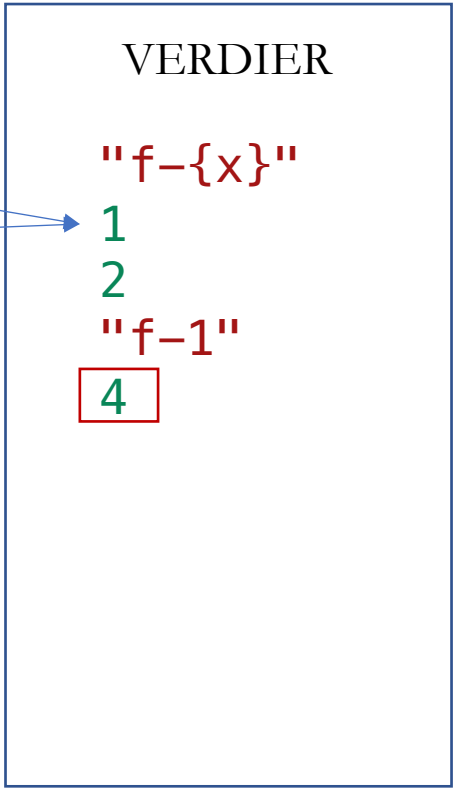
KODESPORING

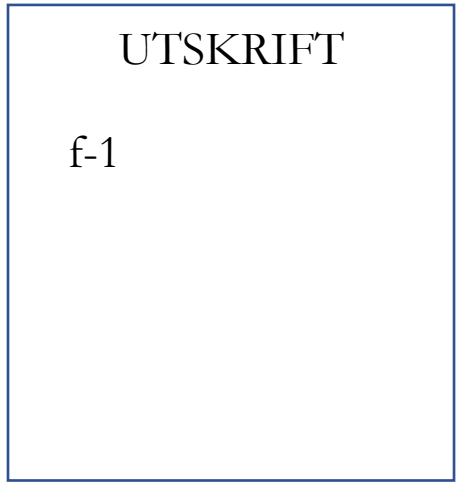
```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```





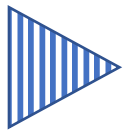


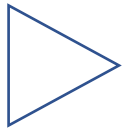


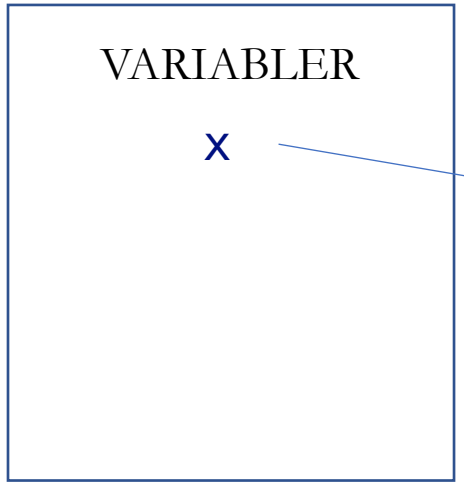


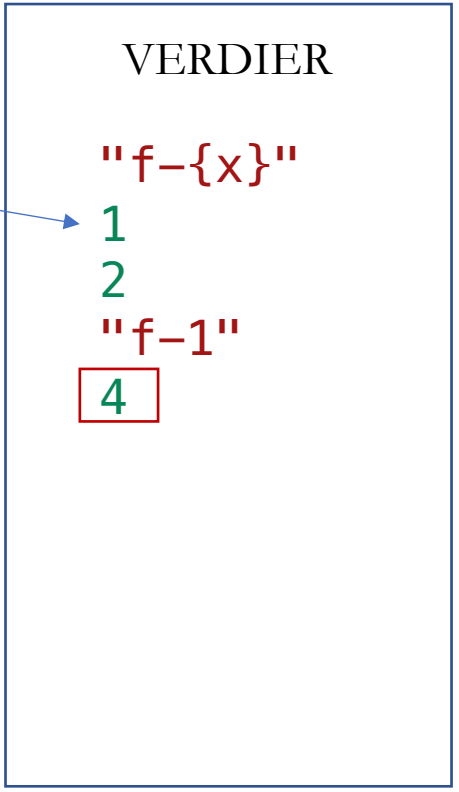
KODESPORING

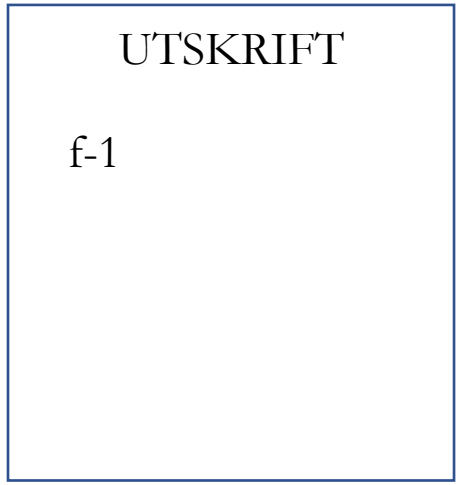
```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```





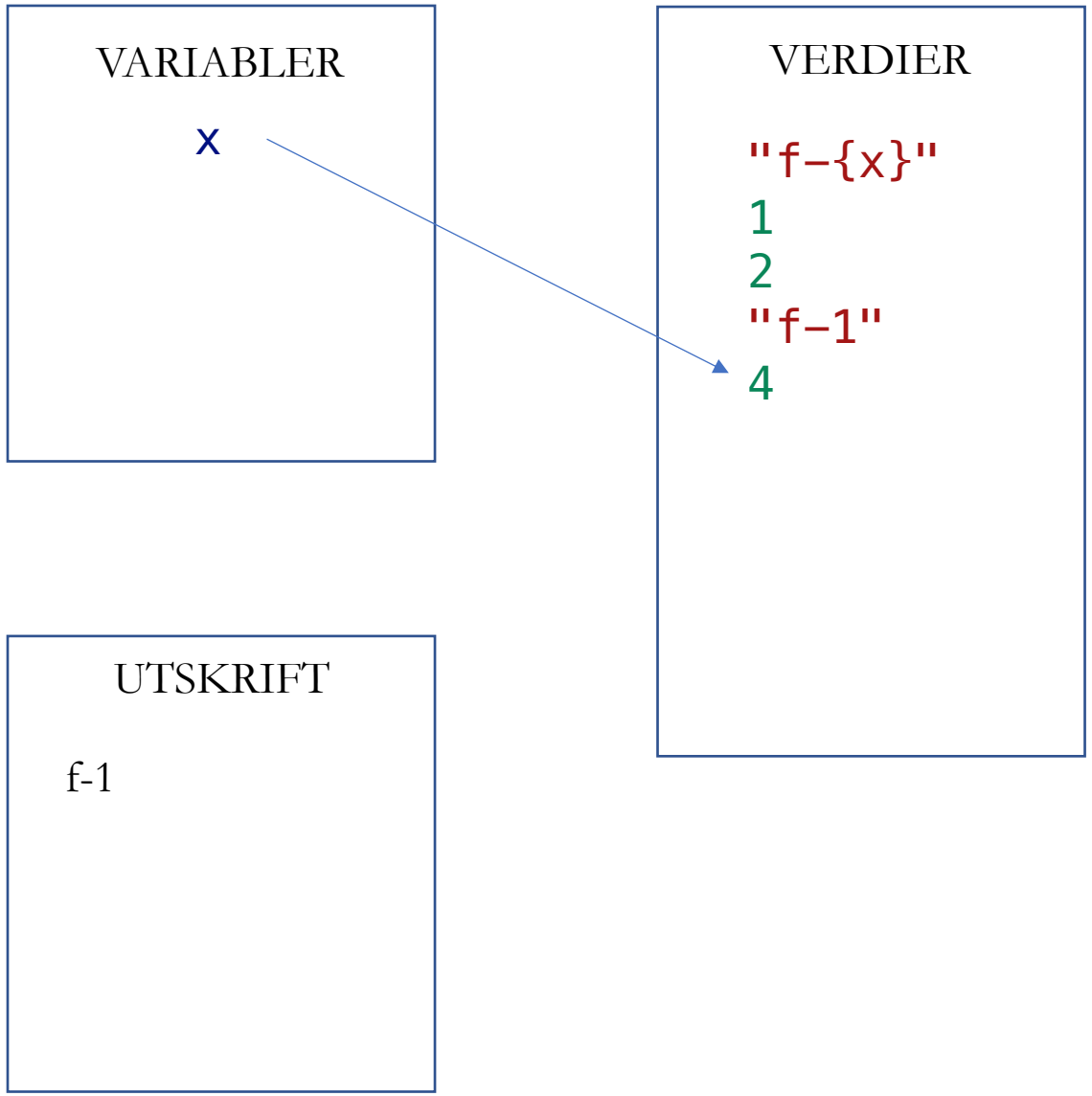






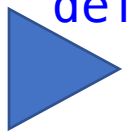
KODESPORING

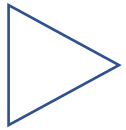
```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```

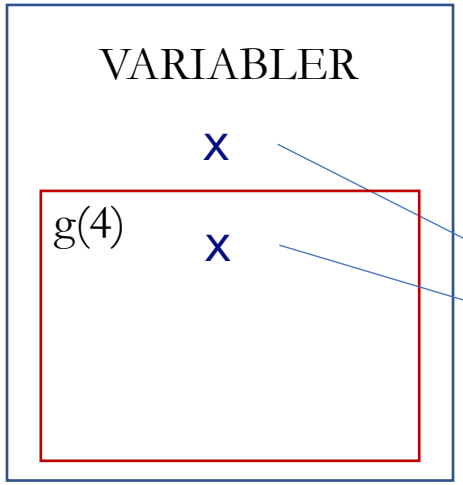


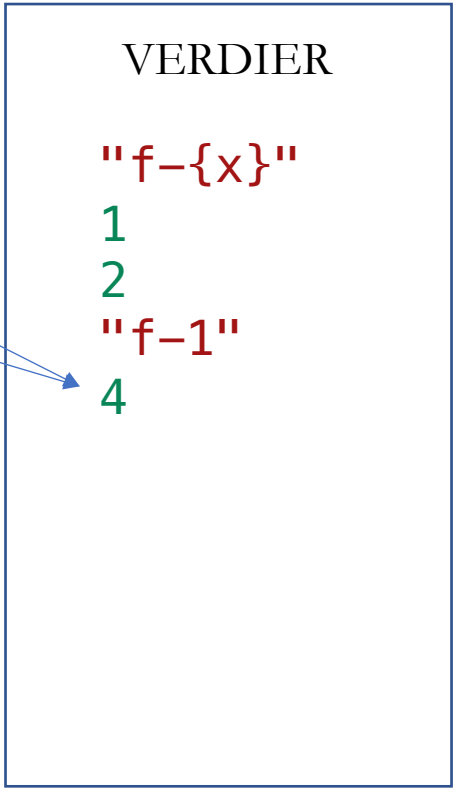
KODESPORING

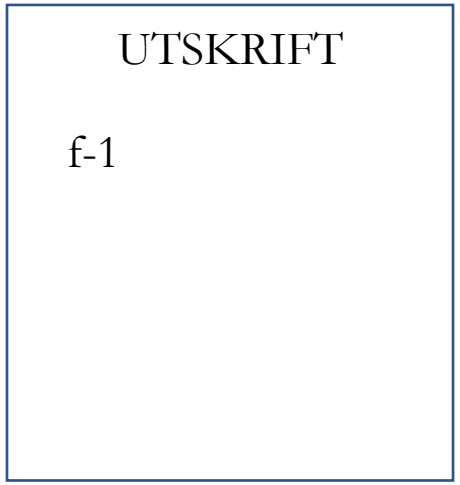
```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```





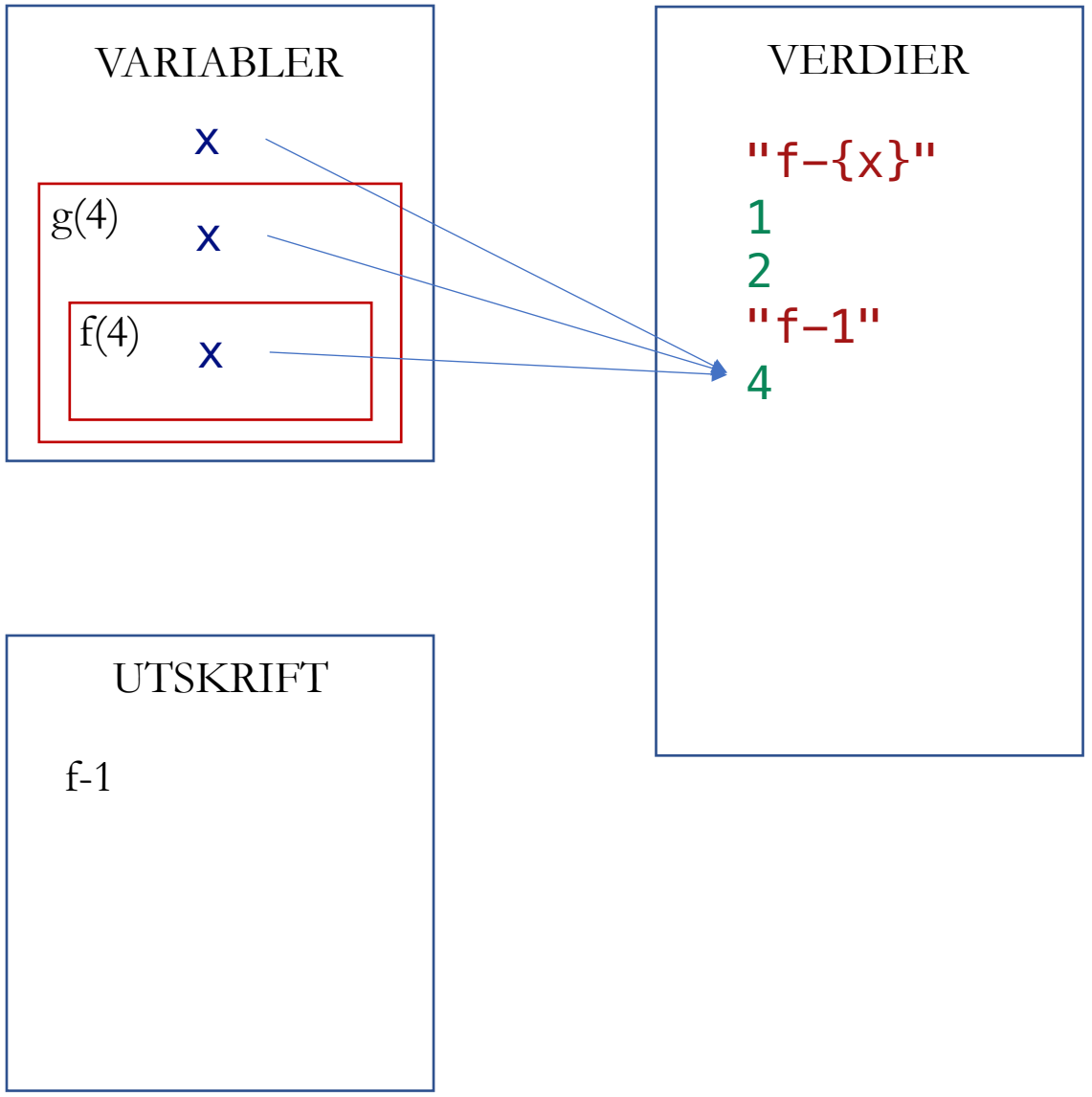






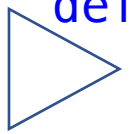
KODESPORING

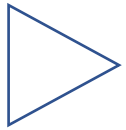
```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```

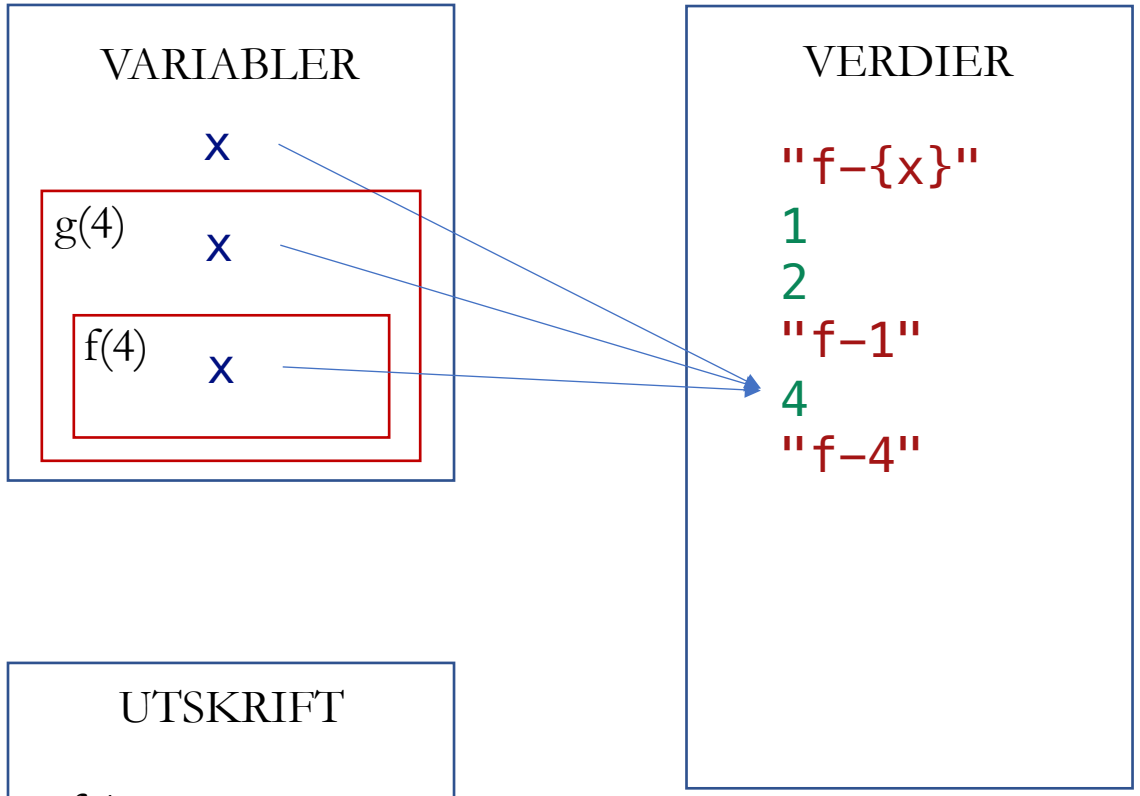


KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```





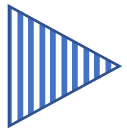


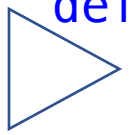
UTSKRIFT

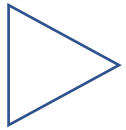
f-1
f-4

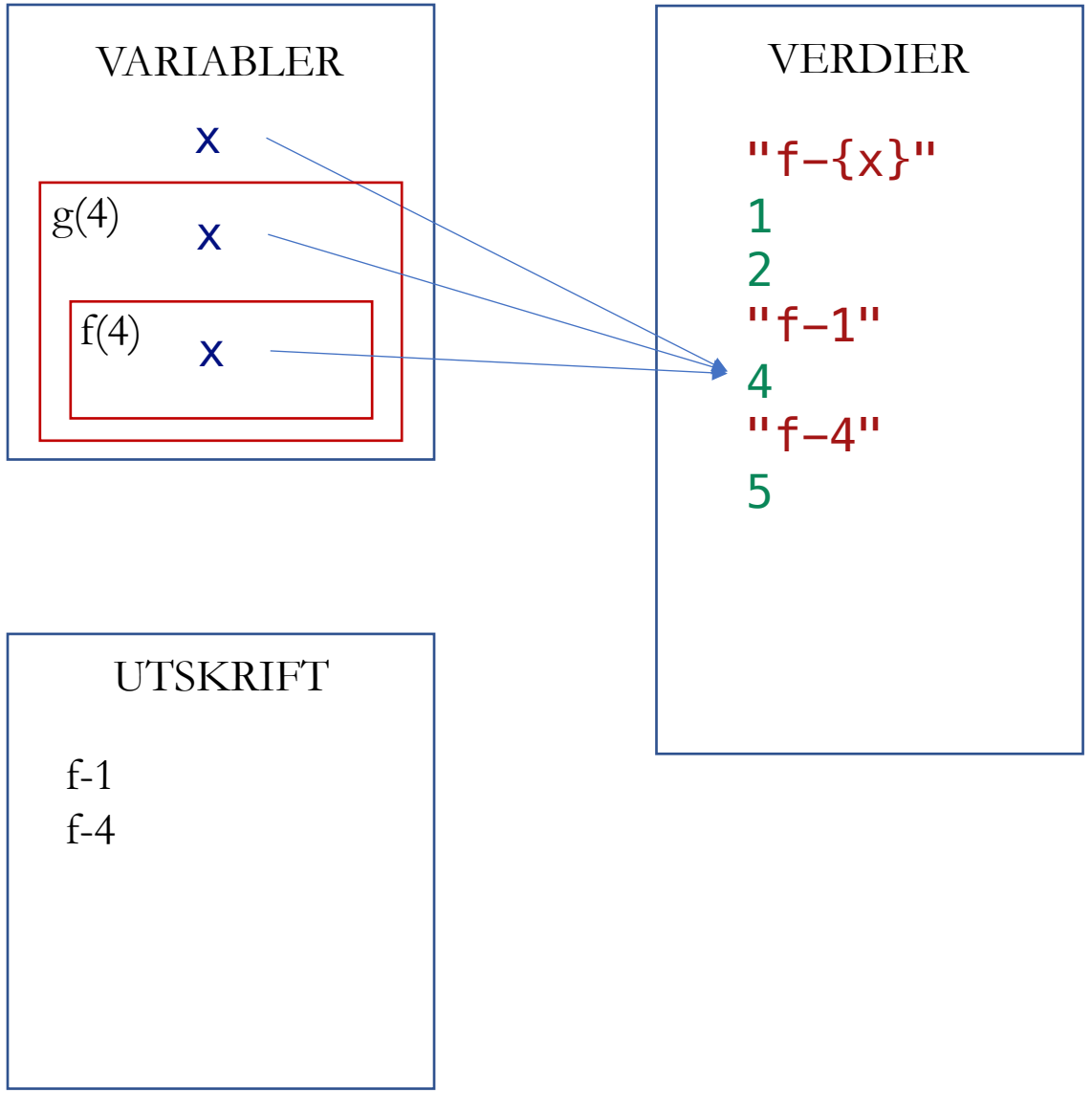
KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```



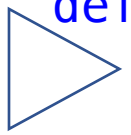


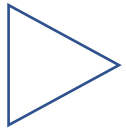


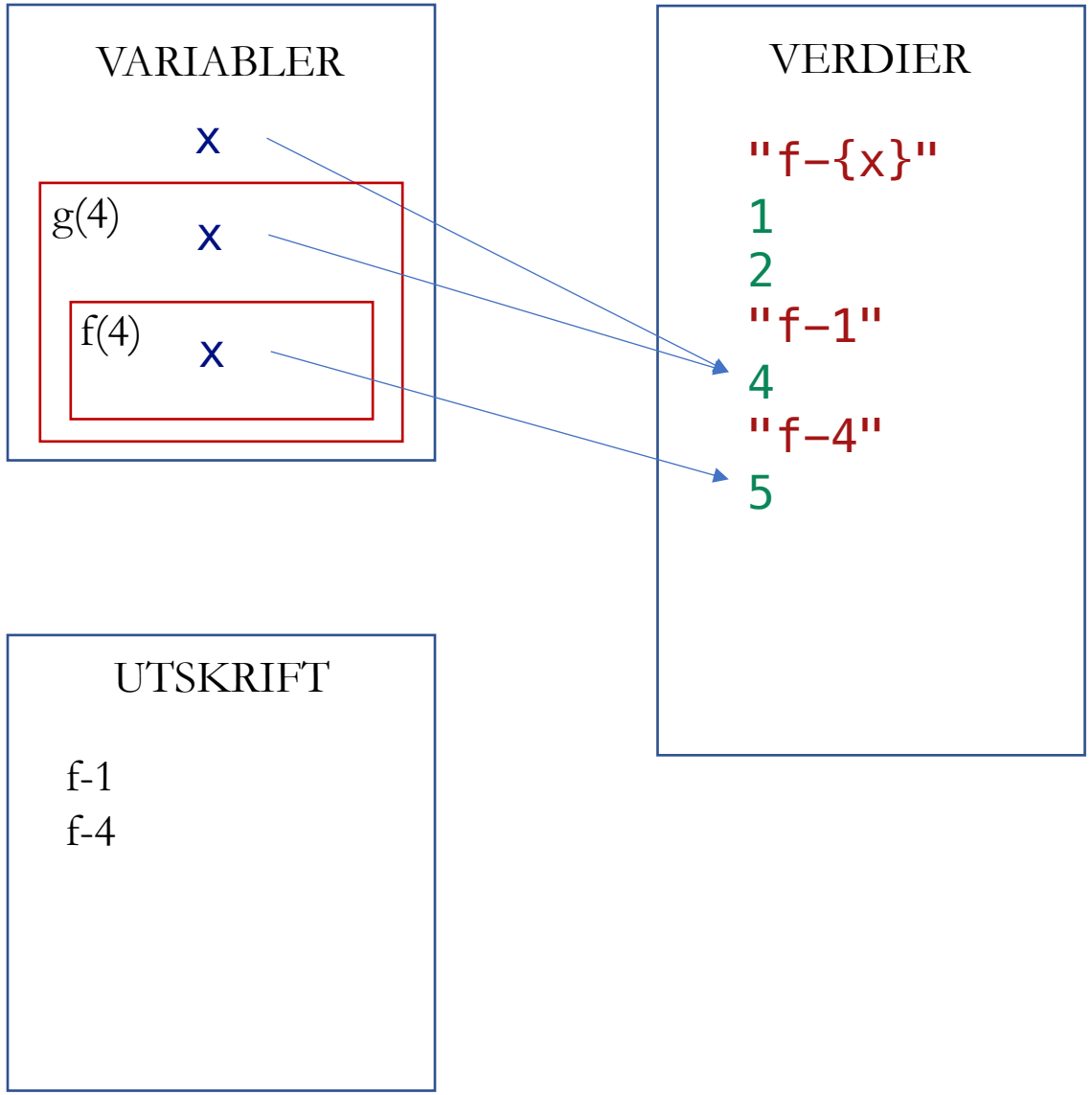


KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```

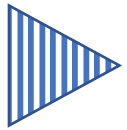


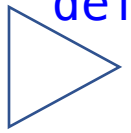


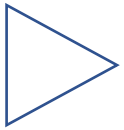


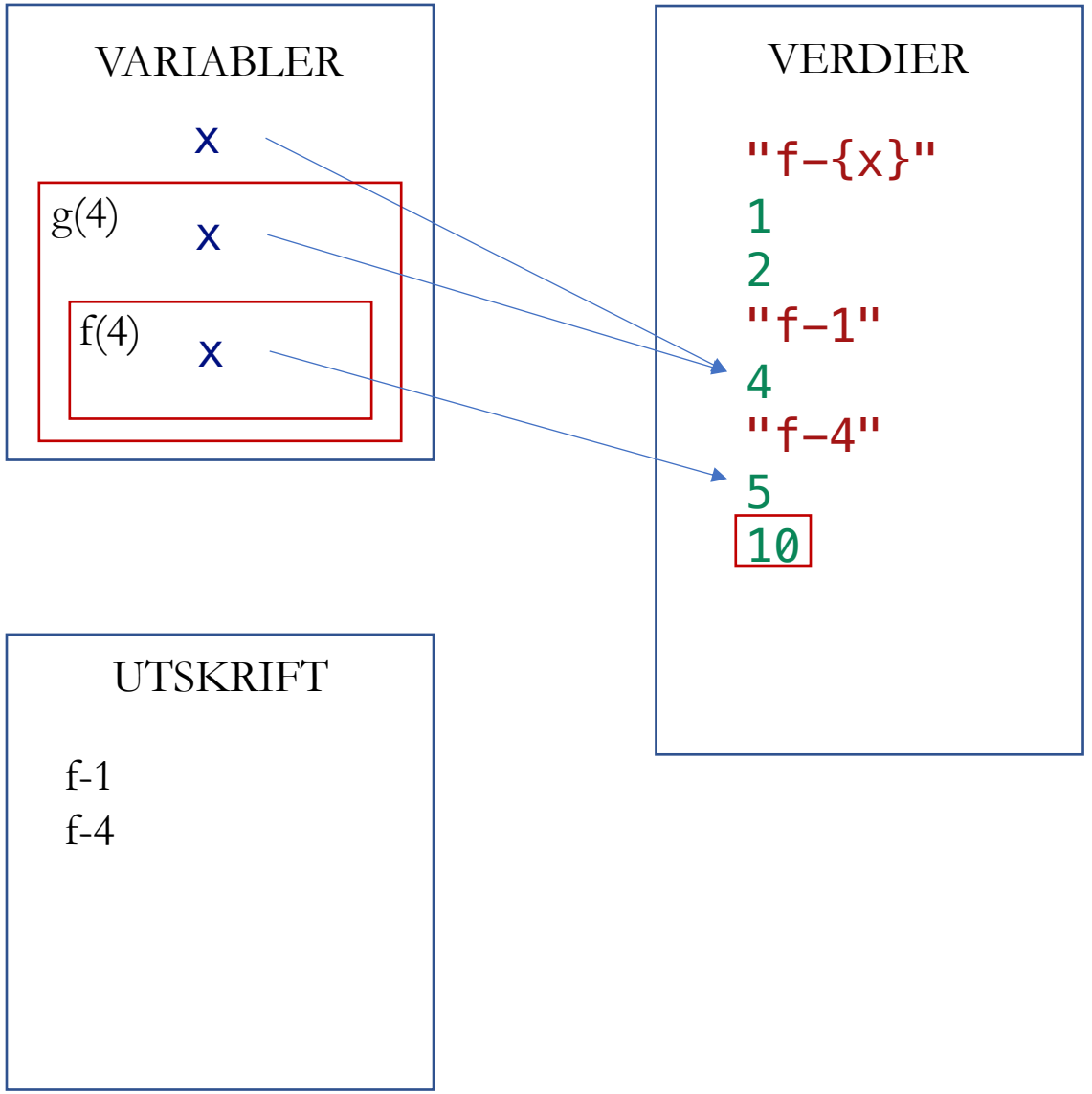
KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```



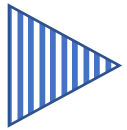


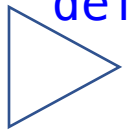


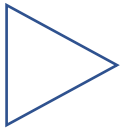


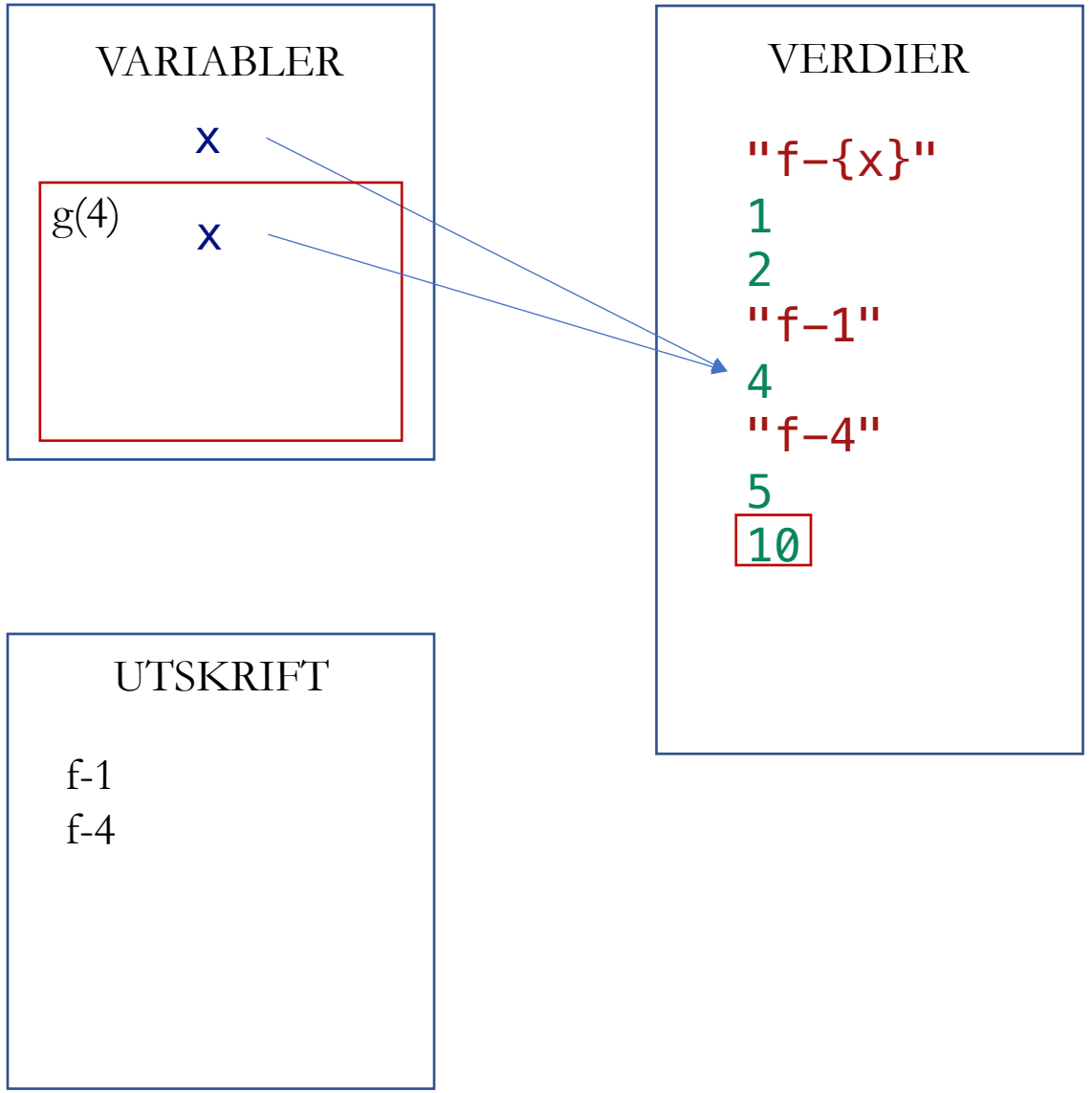
KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```



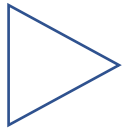


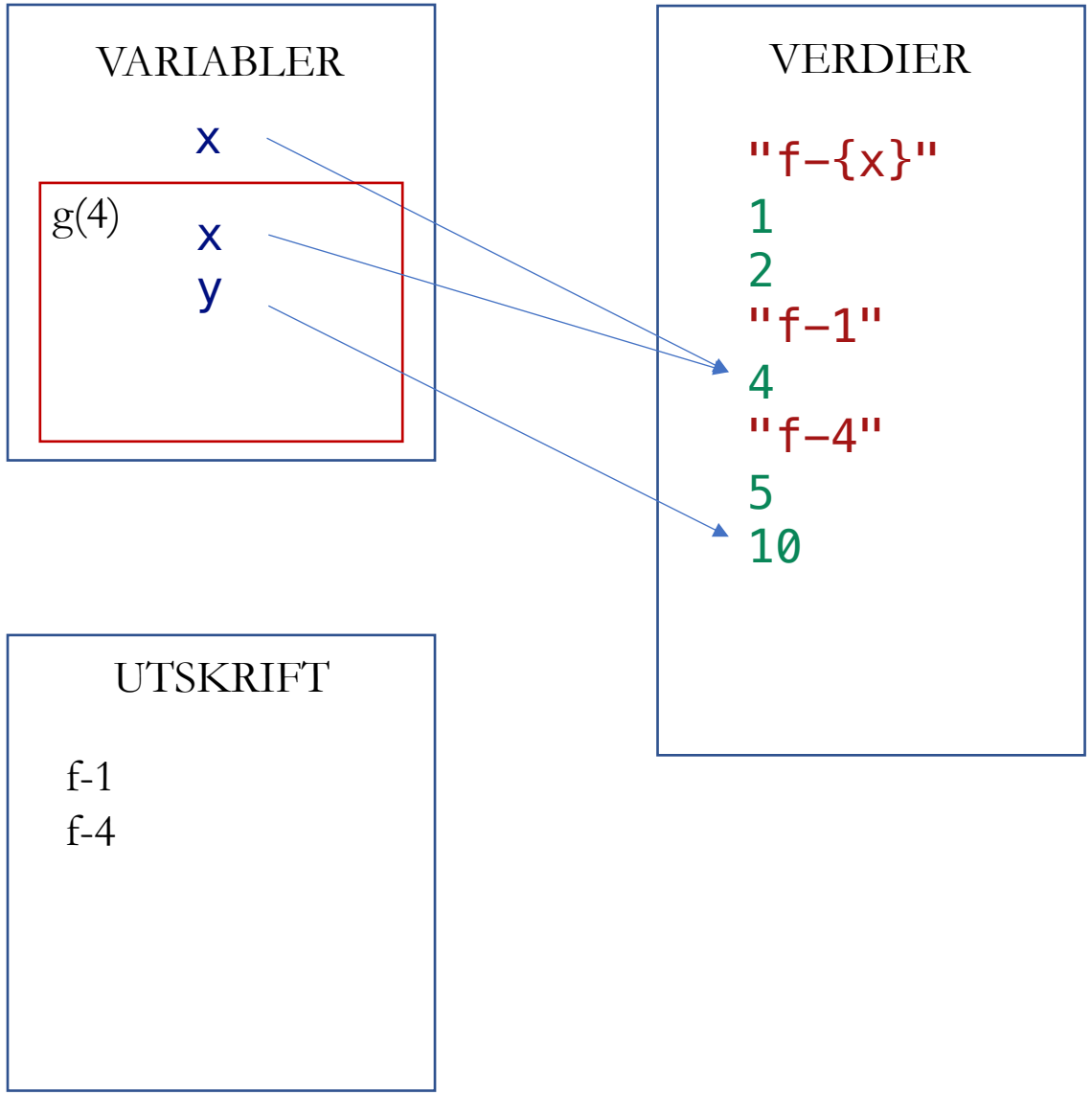




KODESPORING

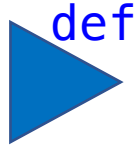
```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```

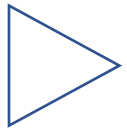


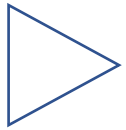


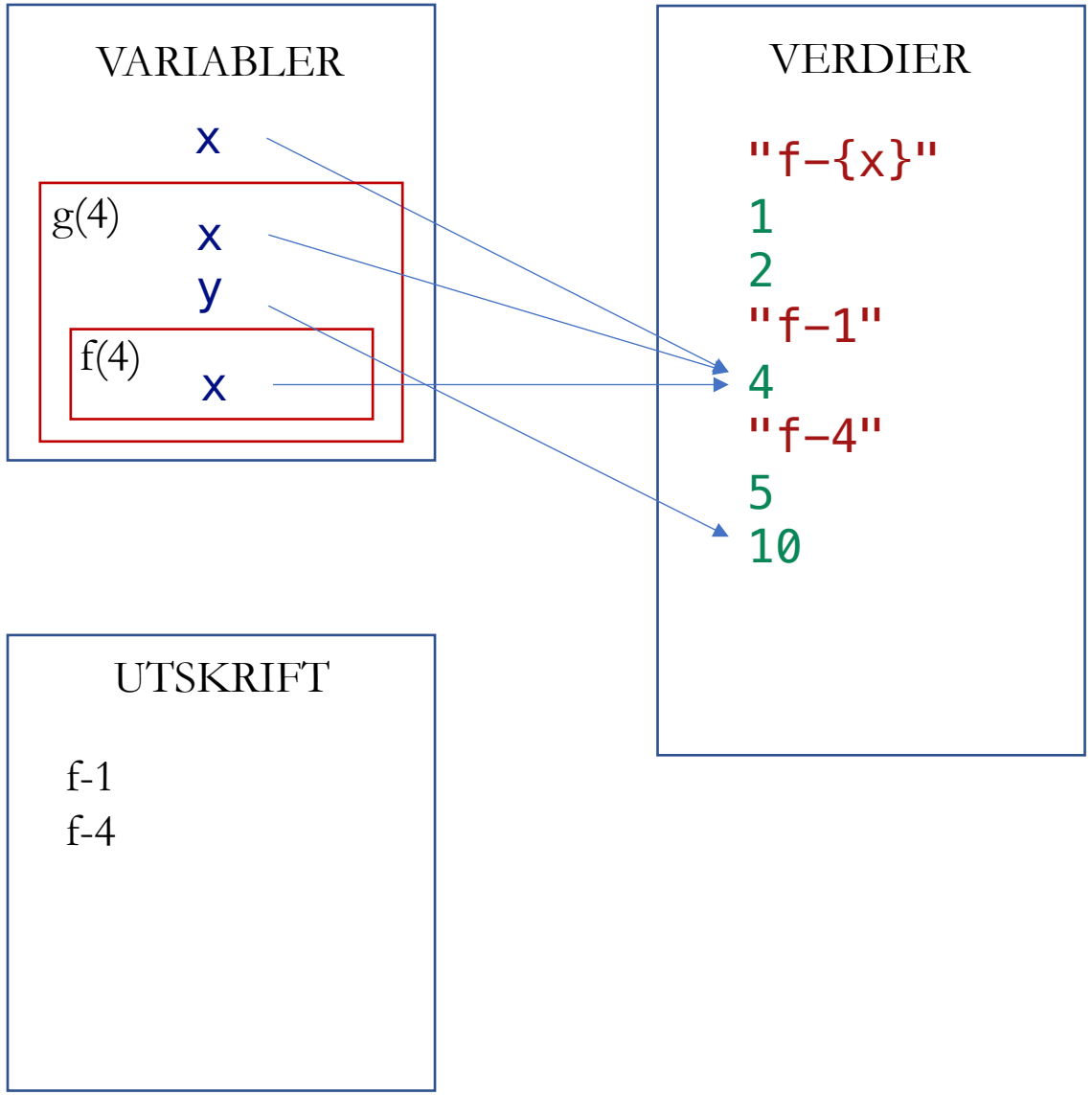
KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```



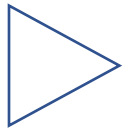


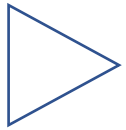


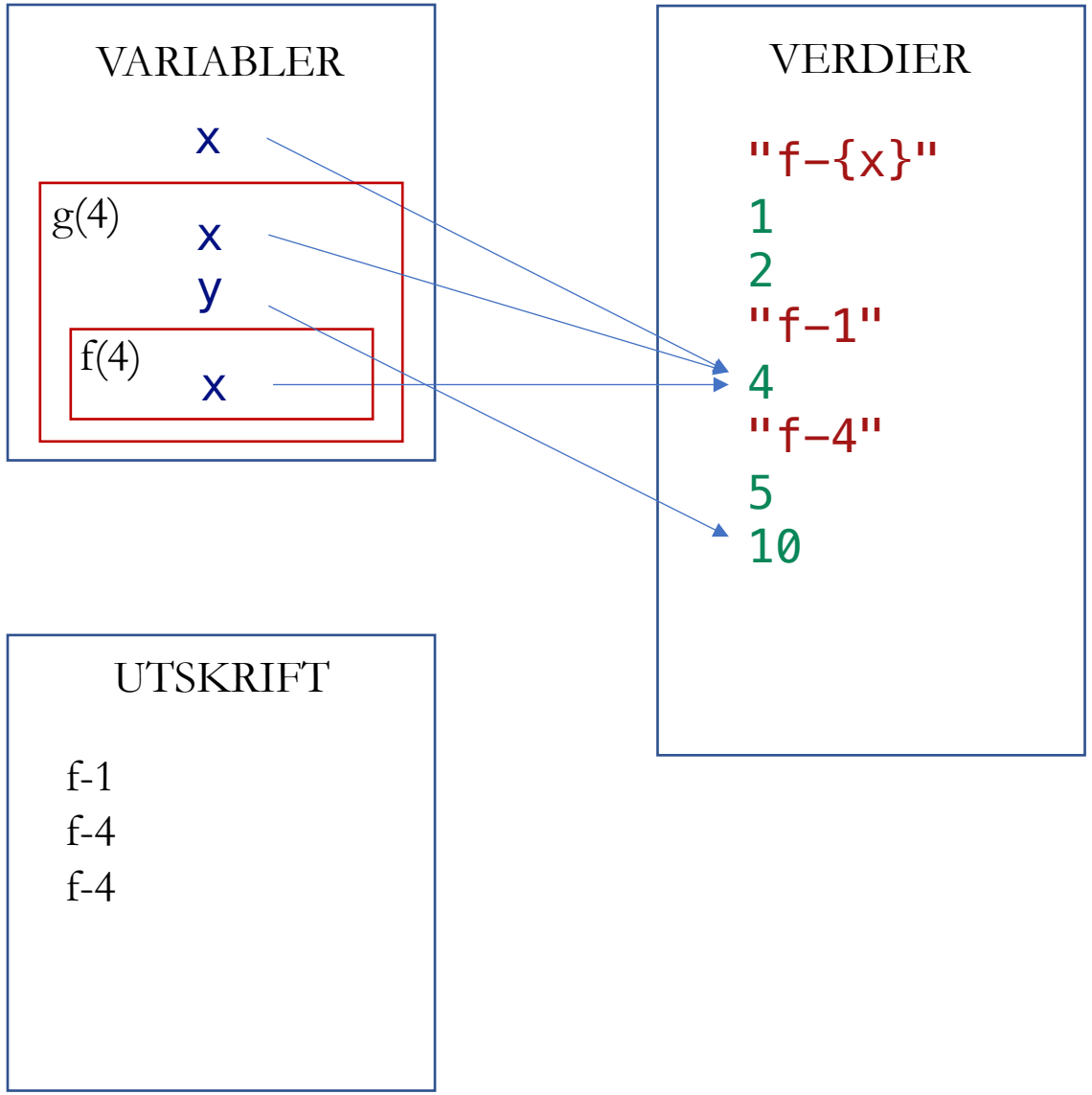


KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```

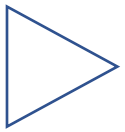


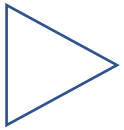


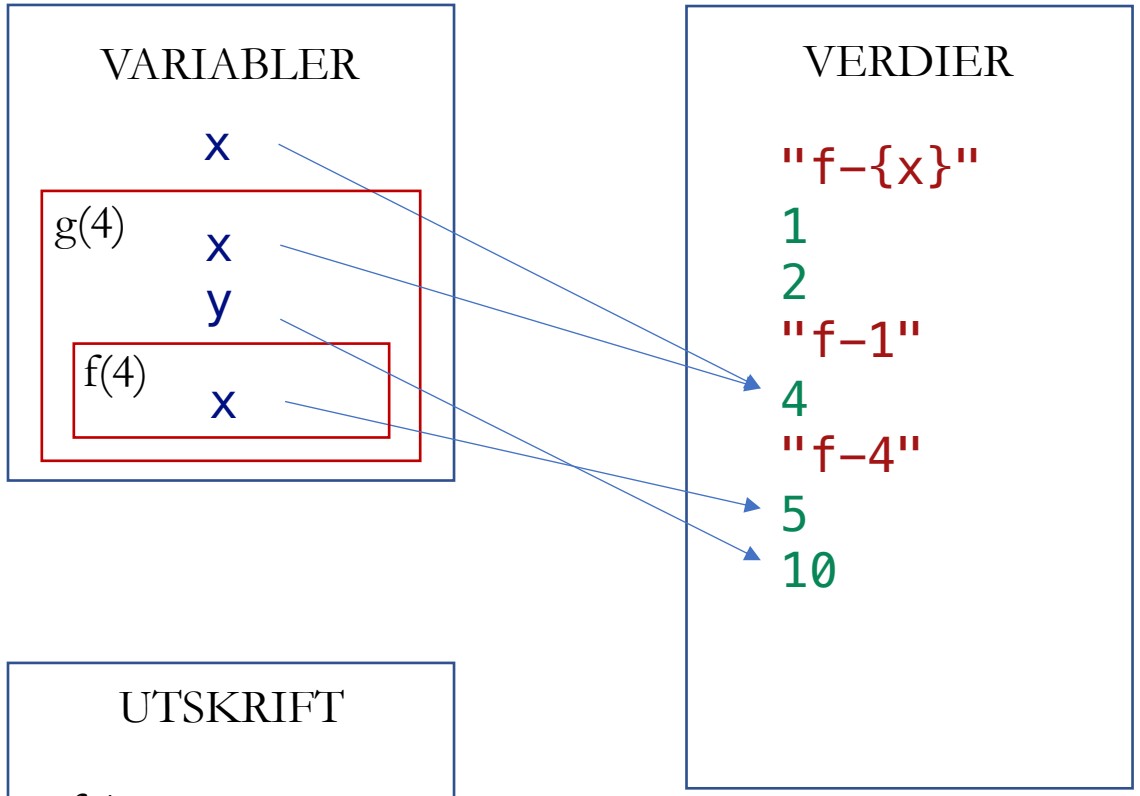


KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```





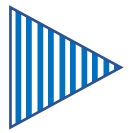


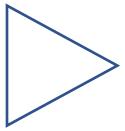
UTSKRIFT

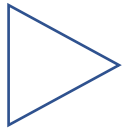
f-1
f-4
f-4

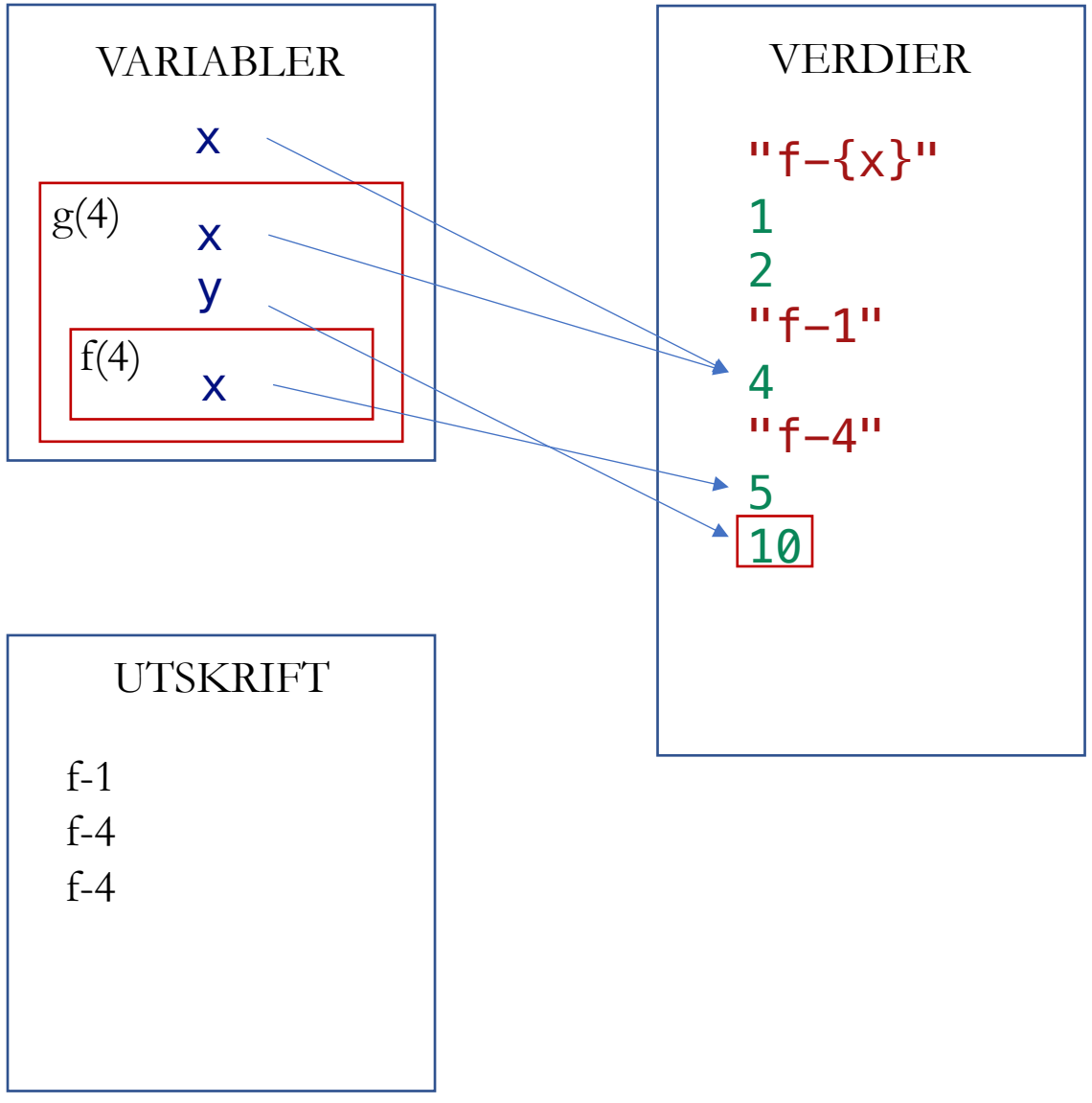
KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```



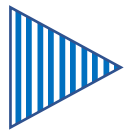


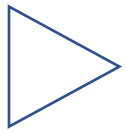


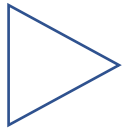


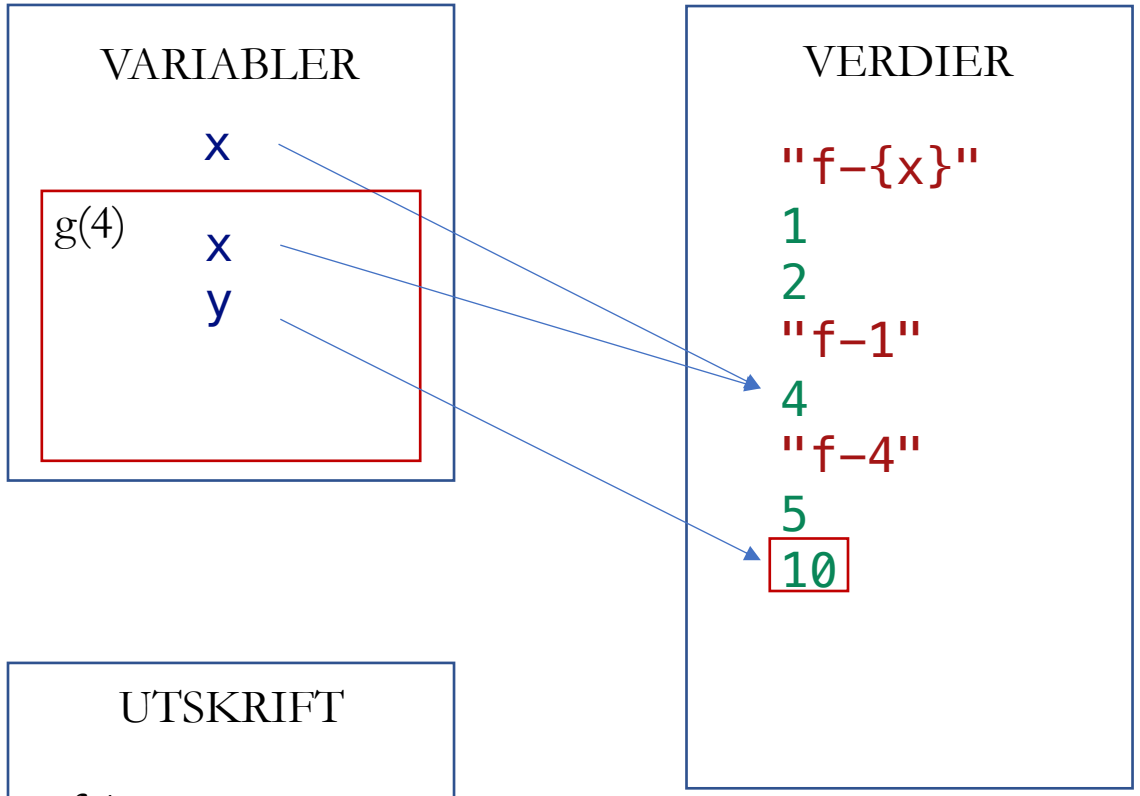
KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```







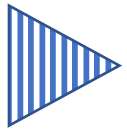


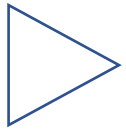
UTSKRIFT

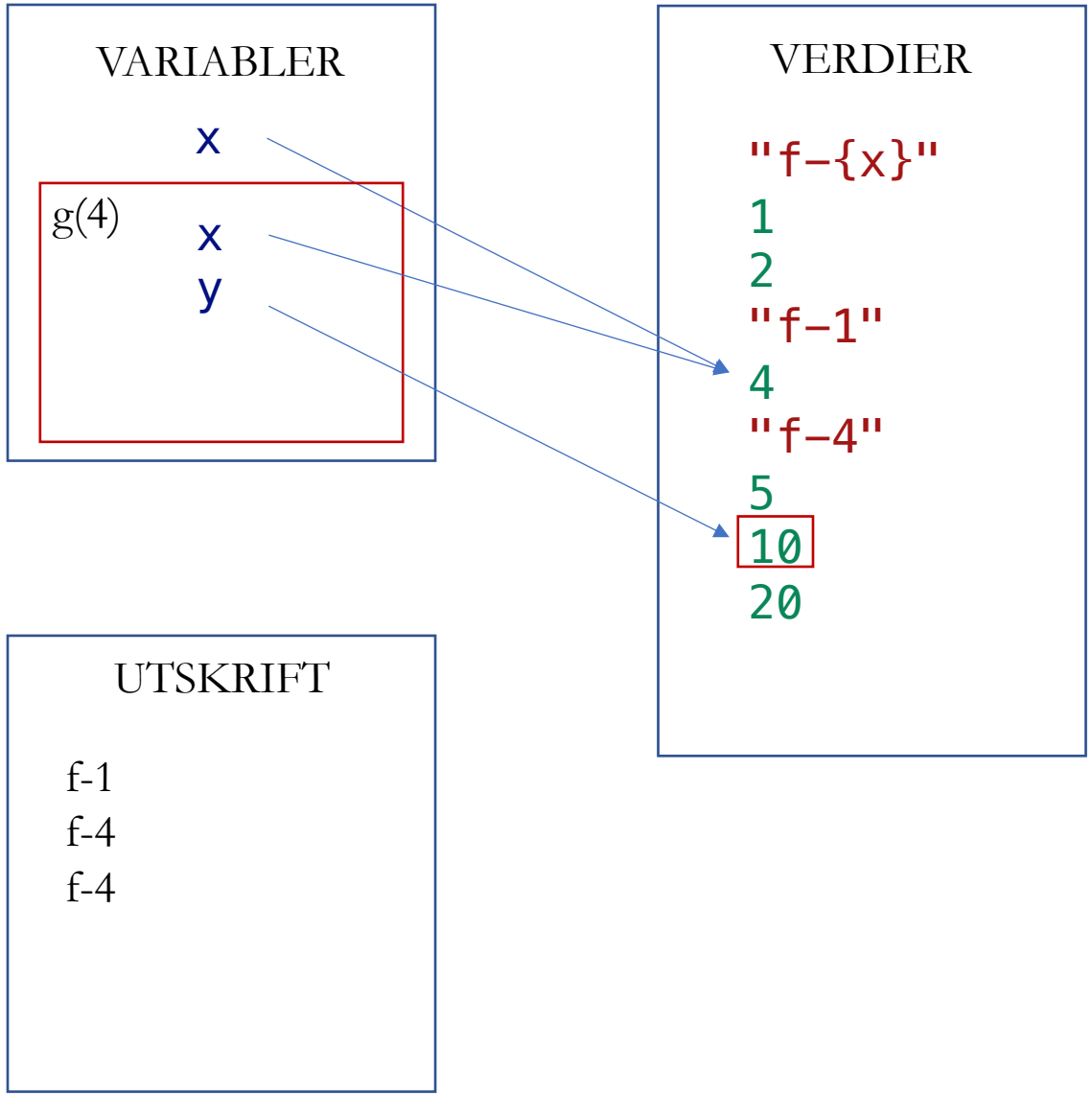
f-1
f-4
f-4

KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```

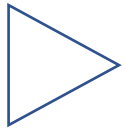


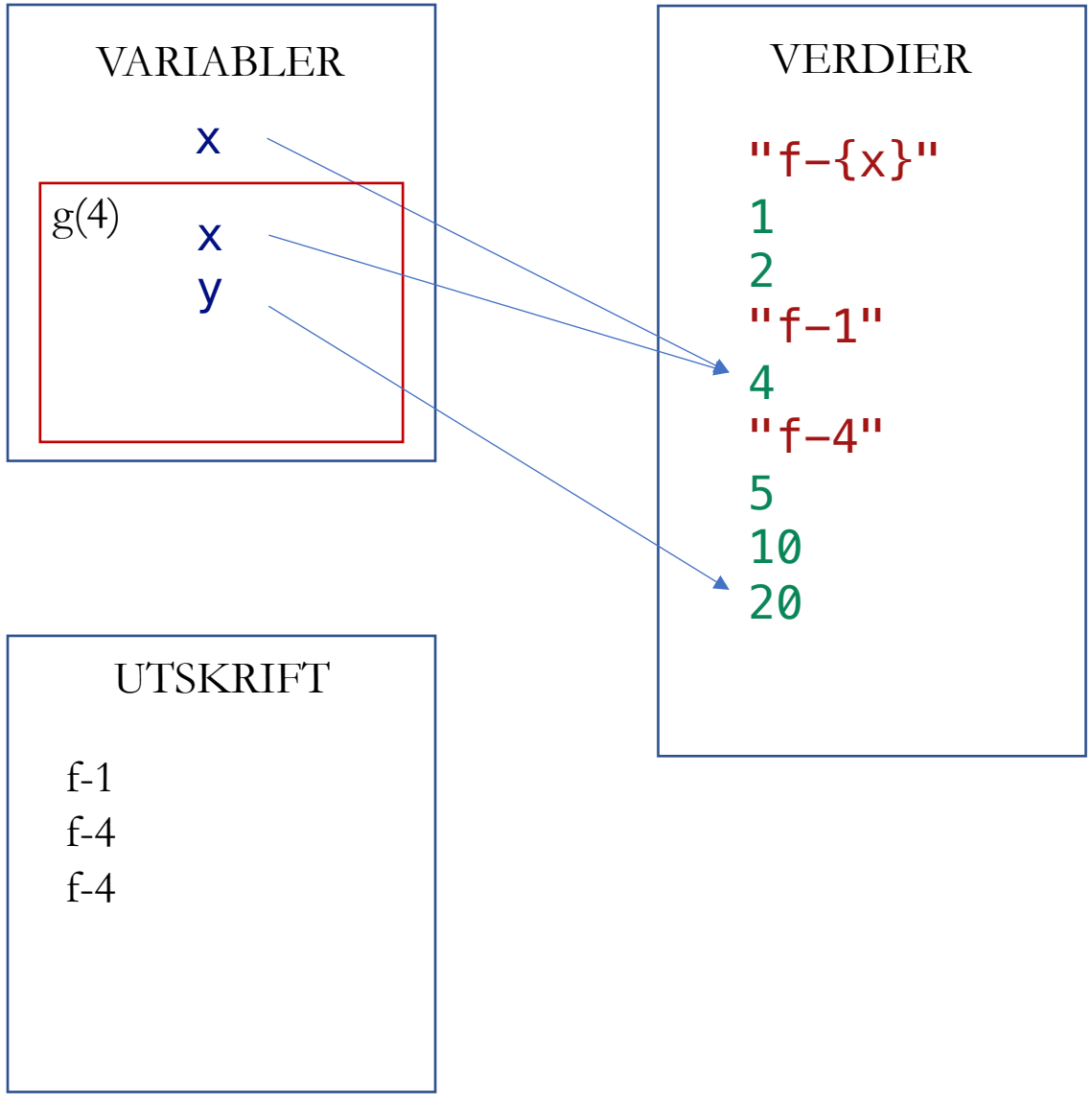




KODESPORING

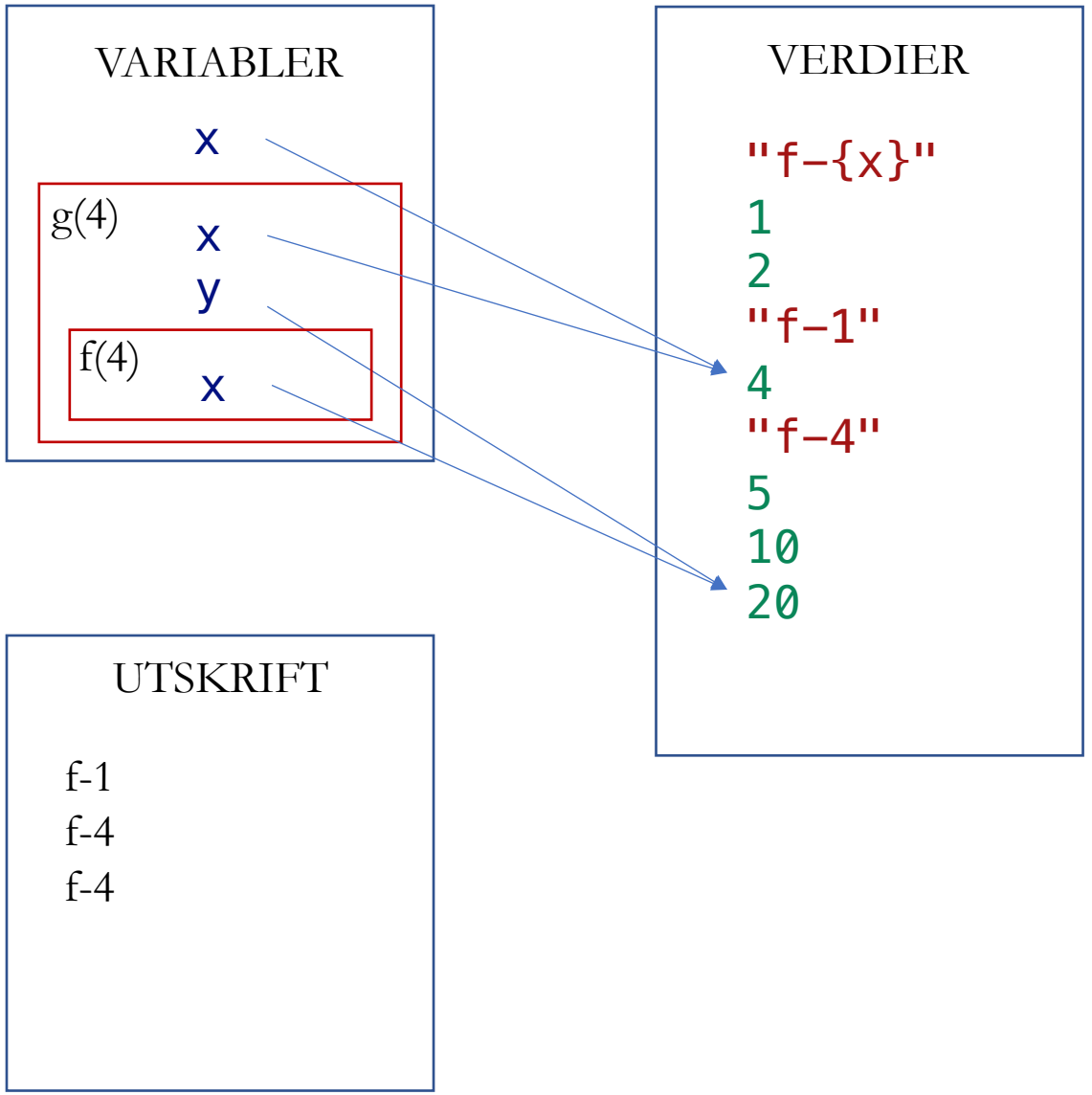
```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```





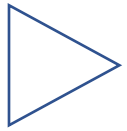
KODESPORING

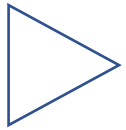
```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```

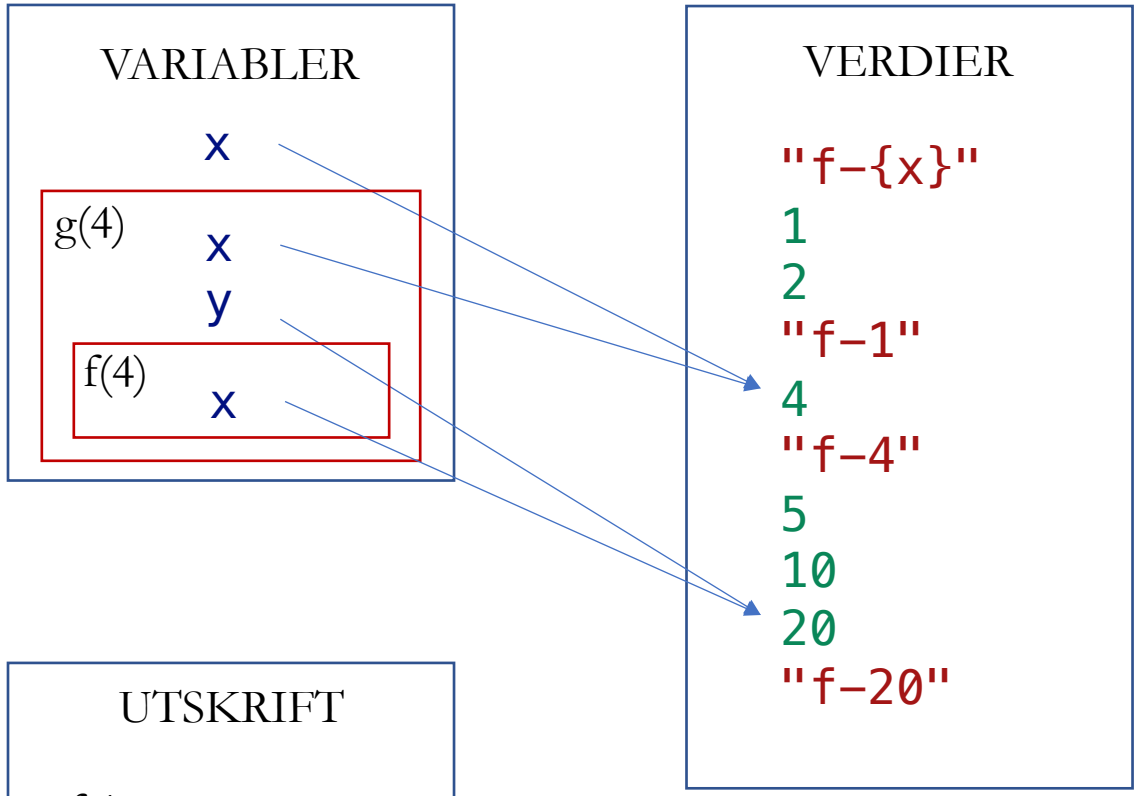


KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```





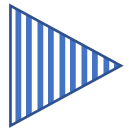


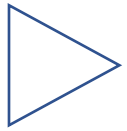
UTSKRIFT

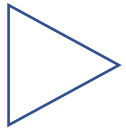
f-1
f-4
f-4
f-20

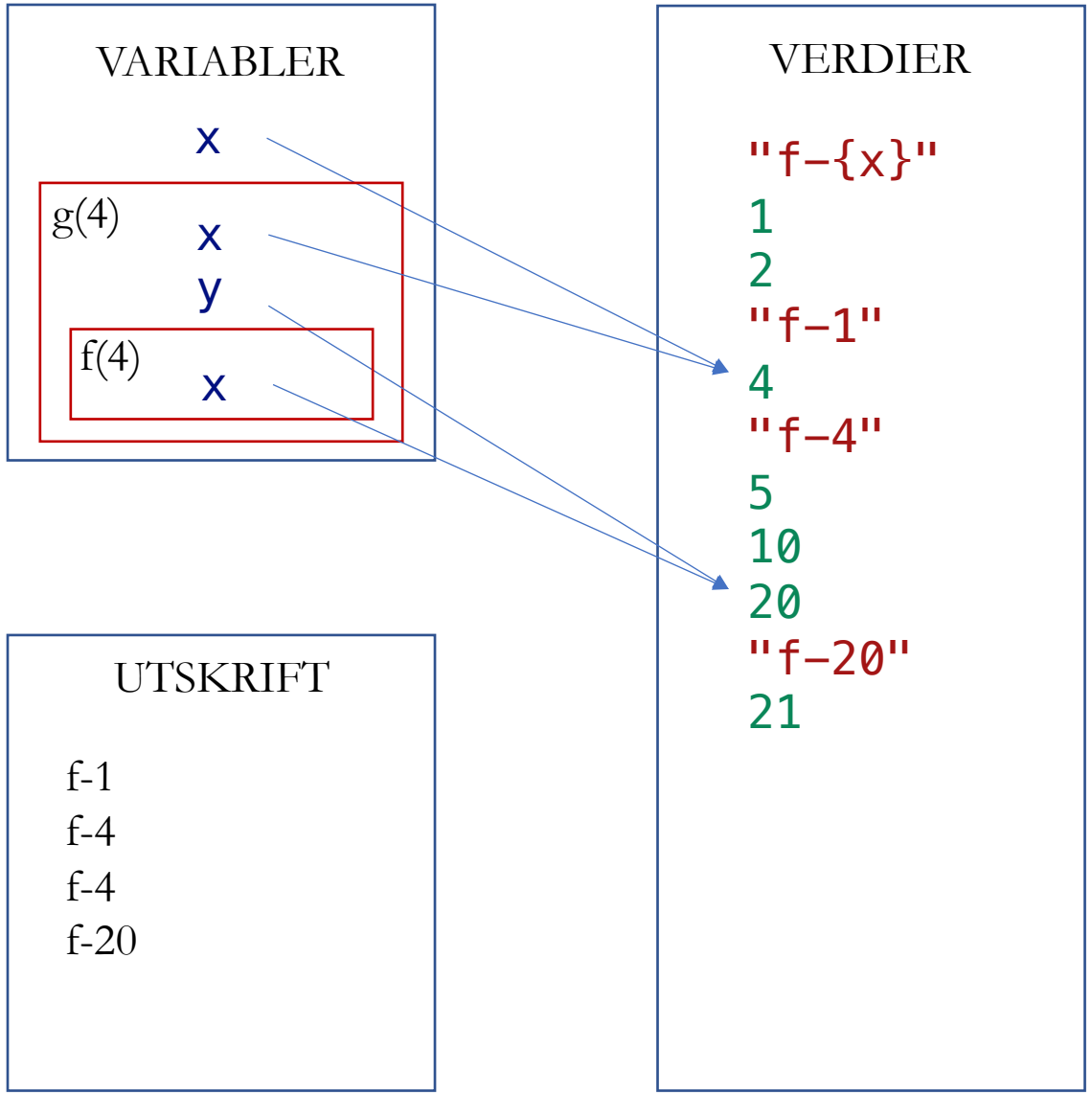
KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```



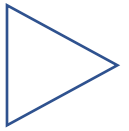


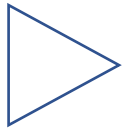


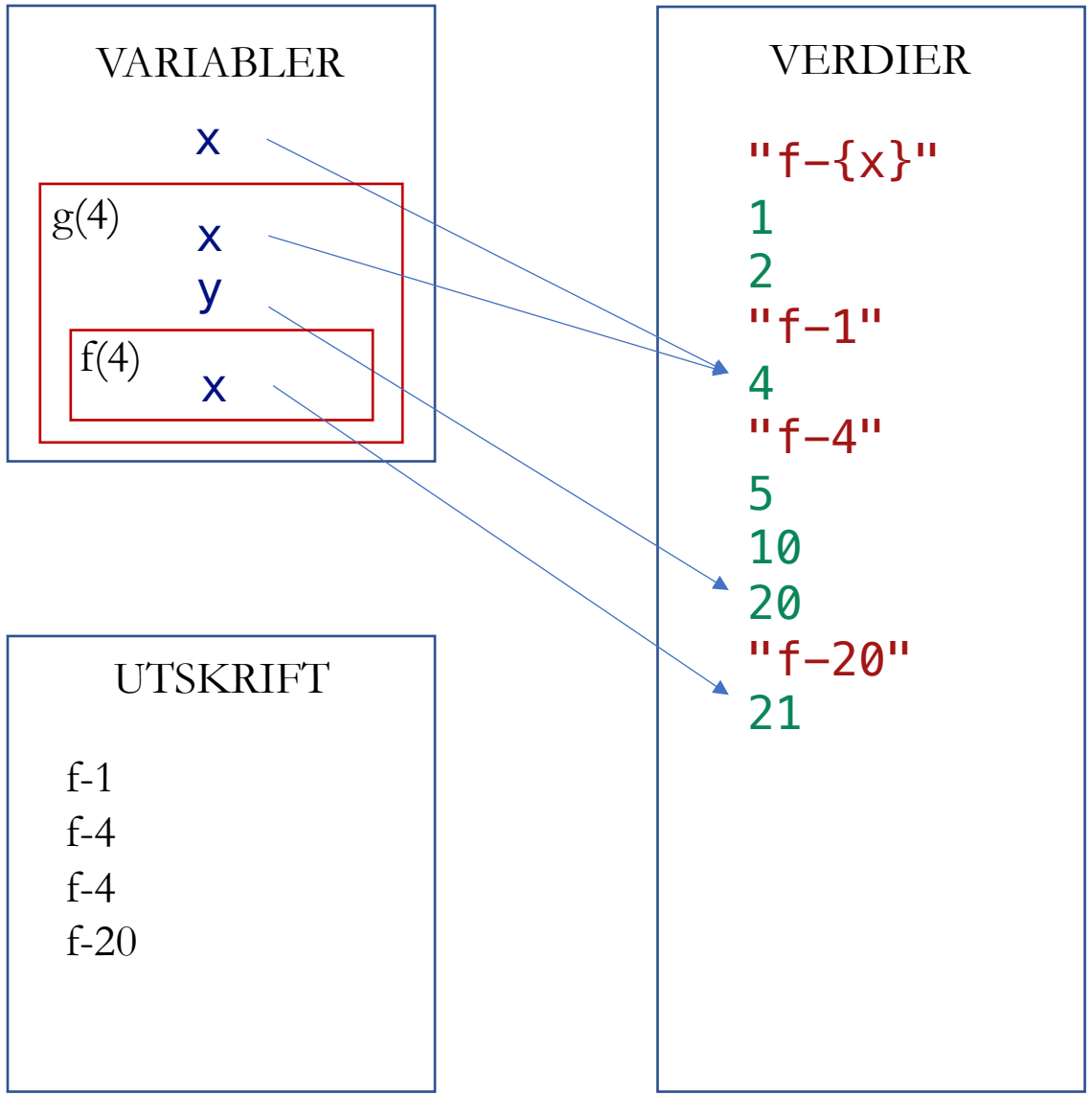


KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```

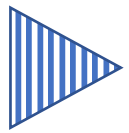


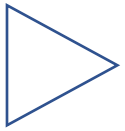


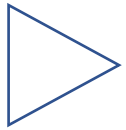


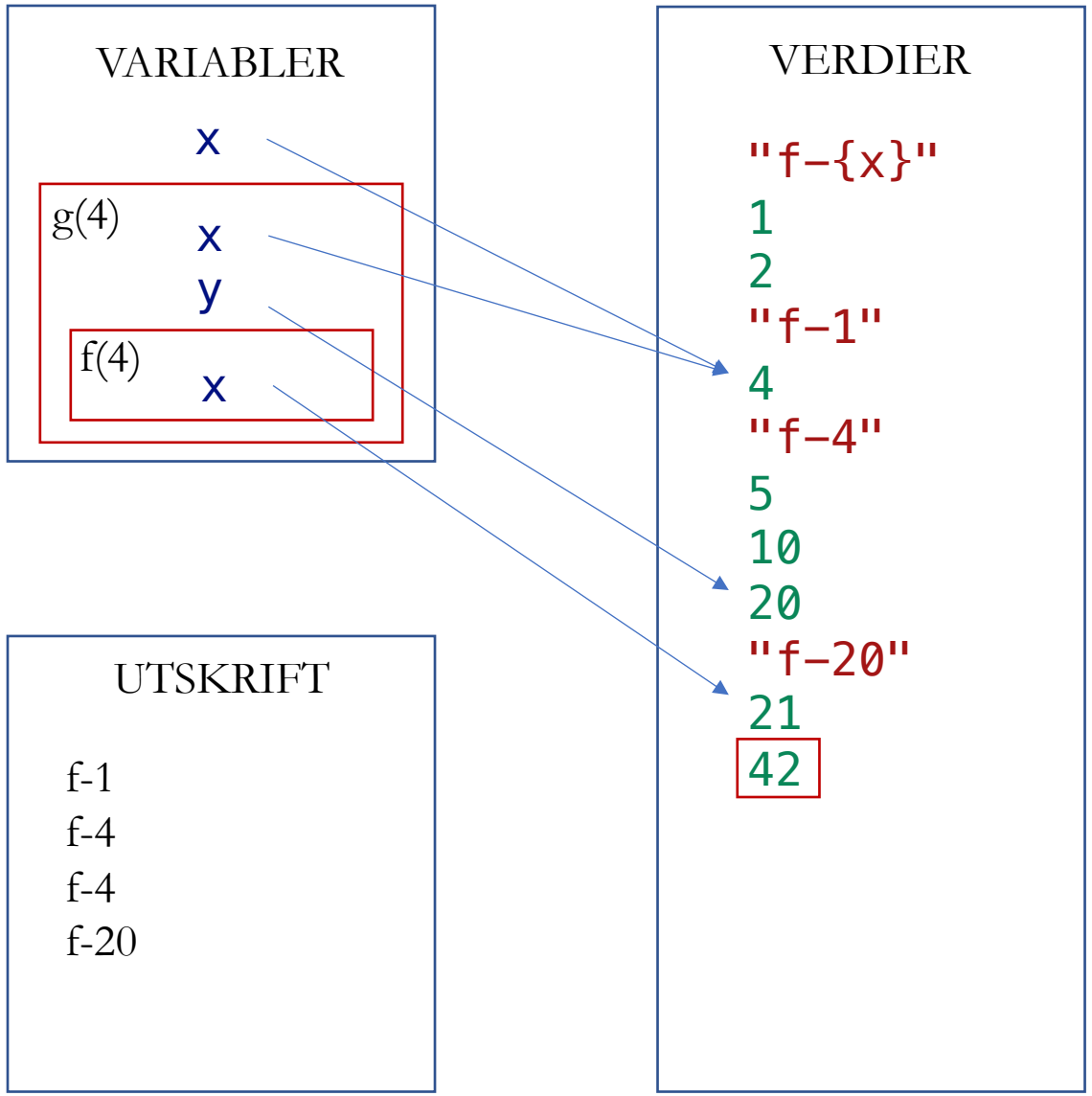
KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```



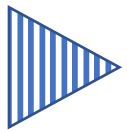


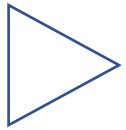


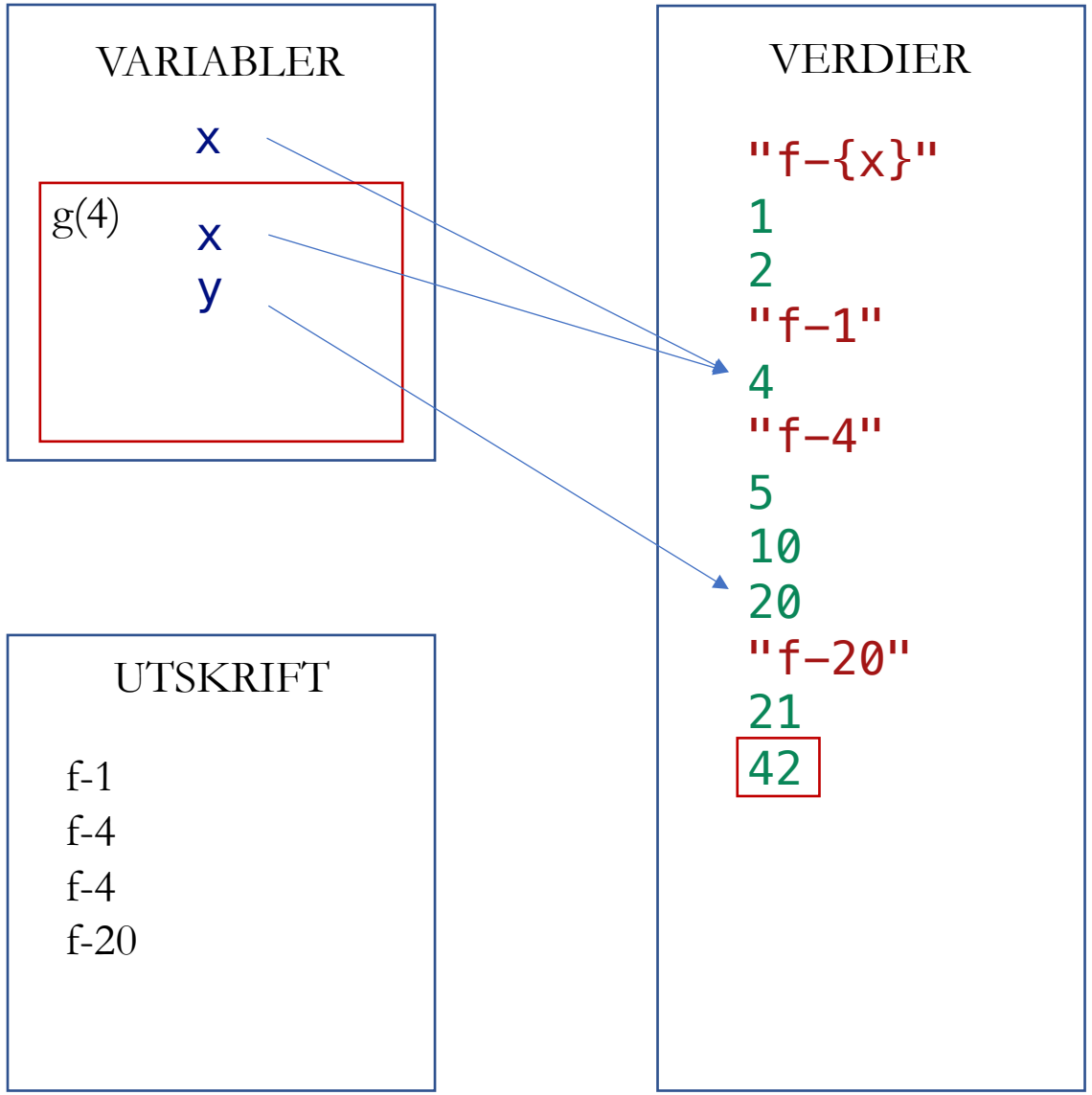


KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```

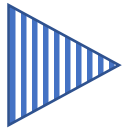






KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```



VARIABLER

x

UTSKRIFT

f-1
f-4
f-4
f-20

VERDIER

"f-{x}"
1
2
"f-1"
4
"f-4"
5
10
20
"f-20"
21
42

KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```

VARIABLER

x

UTSKRIFT

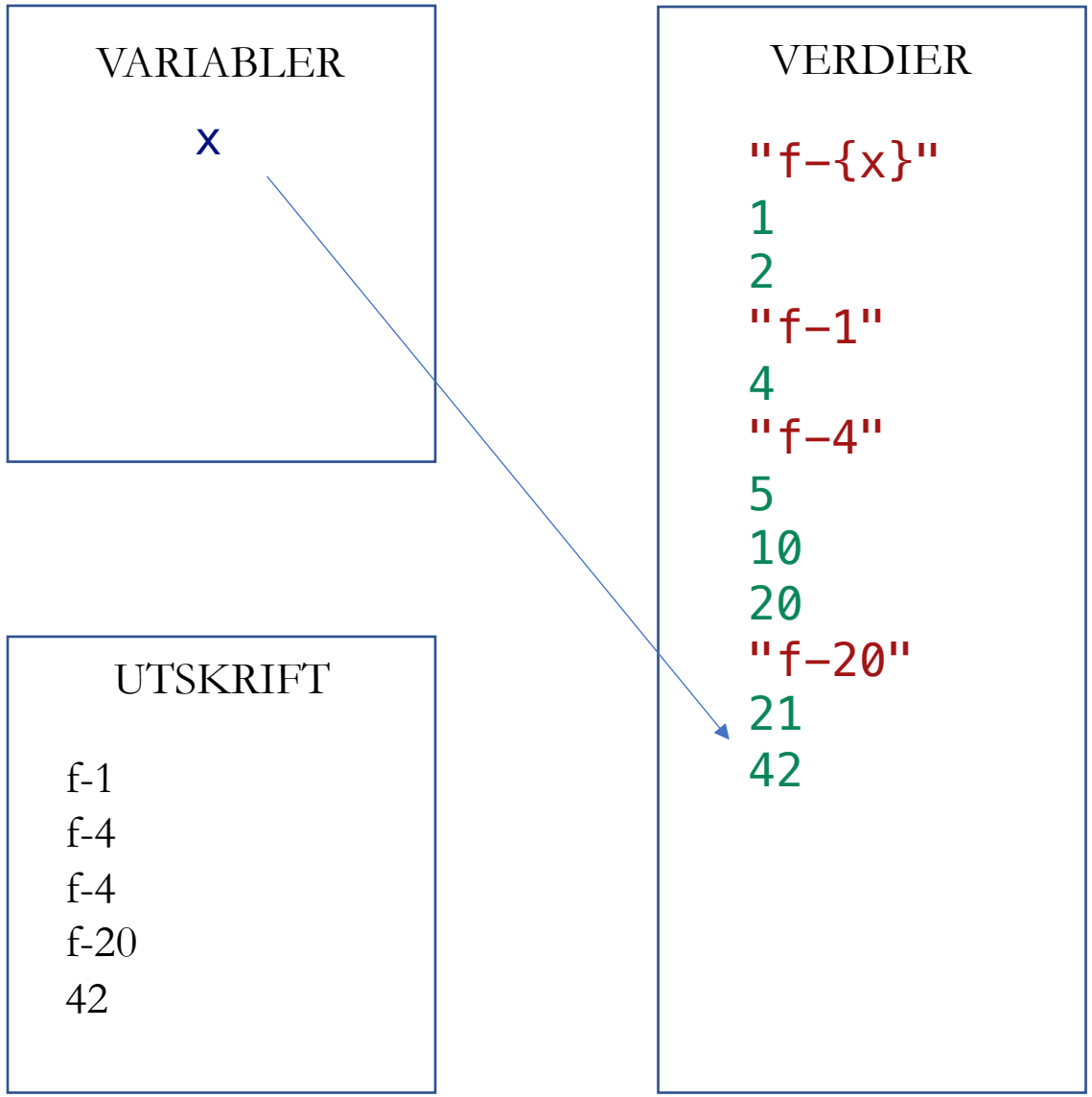
f-1
f-4
f-4
f-20

VERDIER

"f-{x}"
1
2
"f-1"
4
"f-4"
5
10
20
"f-20"
21
42

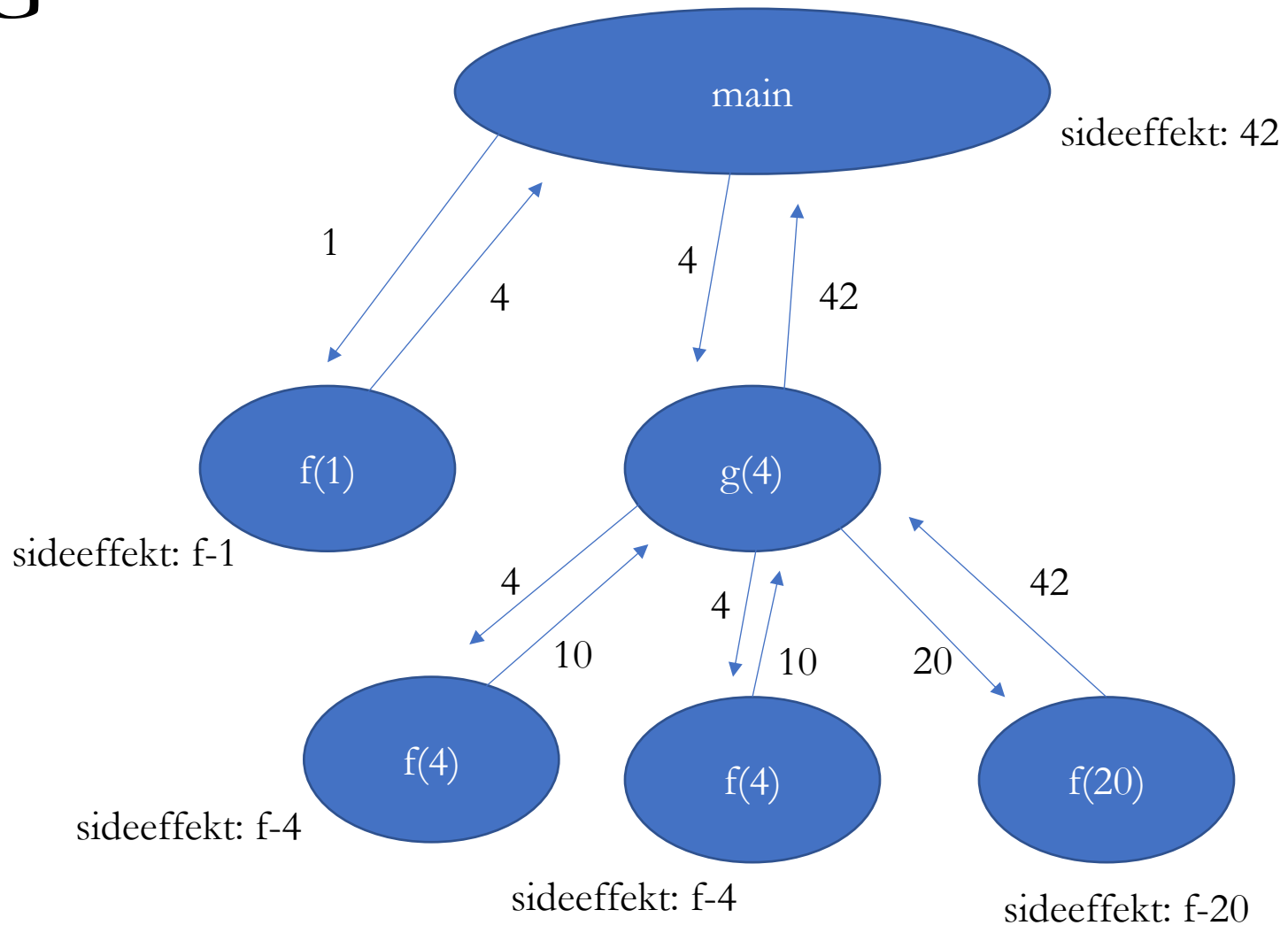
KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```



KODESPORING

```
def f(x):
    print(f"f-{x}")
    x += 1
    return 2*x

def g(x):
    y = f(x)
    y += f(x)
    return f(y)

x = 1
x = f(x)
x = g(x)
print(x)
```



UNIVERSITETET I BERGEN

FEIL

INF100

HØST 2022

Torstein Strømme

TRE FORMER FOR FEIL

- Syntaks
  - Programmet krasjer før det begynner å kjøre

```
def volume_of_box(x, y, z)
    print(x * y + z)

print("Det er plass til " + volum_of_box(1, 2, 3) + " m3 i boksen
```

```
line 4
    print("Det er plass til " + volum_of_box(1, 2, 3) + " m3 i boksen
                                                      ^
SyntaxError: unterminated string literal (detected at line 4)
```

TRE FORMER FOR FEIL

- Syntaks
  - Programmet krasjer før det begynner å kjøre

```
def volume_of_box(x, y, z)
    print(x * y + z)

print("Det er plass til " + volum_of_box(1, 2, 3) + " m3 i boksen"
```

```
line 1
    def volume_of_box(x, y, z)
                              ^
SyntaxError: expected ':'
```

TRE FORMER FOR FEIL

- Syntaks
  - Programmet krasjer før det begynner å kjøre

```
def volume_of_box(x, y, z):
    print(x * y + z)

print("Det er plass til " + volum_of_box(1, 2, 3) + " m3 i boksen"
```

```
line 4
    print("Det er plass til " + volum_of_box(1, 2, 3) + " m3 i boksen"
          ^
SyntaxError: '(' was never closed
```

TRE FORMER FOR FEIL

- Krasj (engelsk: runtime error)
  - Programmet krasjer når det kjører

```
def volume_of_box(x, y, z):
    print(x * y + z)

print("Det er plass til " + volum_of_box(1, 2, 3) + " m3 i boksen")
```

```
line 4, in <module>
    print("Det er plass til " + volum_of_box(1, 2, 3) + " m3 i boksen")
NameError: name 'volum_of_box' is not defined. Did you mean:
'volume_of_box'?
```

TRE FORMER FOR FEIL

- Krasj (engelsk: runtime error)
  - Programmet krasjer når det kjører

```
def volume_of_box(x, y, z):
    print(x * y + z)

print("Det er plass til " + volume_of_box(1, 2, 3) + " m3 i boksen")
```

```
line 4, in <module>
    print("Det er plass til " + volume_of_box(1, 2, 3) + " m3 i boksen")
TypeError: can only concatenate str (not "NoneType") to str
```

TRE FORMER FOR FEIL

- Krasj (engelsk: runtime error)
  - Programmet krasjer når det kjører

```
def volume_of_box(x, y, z):
    return x * y + z

print("Det er plass til " + volume_of_box(1, 2, 3) + " m3 i boksen")
```

```
line 4, in <module>
    print("Det er plass til " + volume_of_box(1, 2, 3) + " m3 i boksen")
TypeError: can only concatenate str (not "int") to str
```

TRE FORMER FOR FEIL

- Logisk feil
  - Programmet gir feil svar

```
def volume_of_box(x, y, z):
    return x * y + z

print("Det er plass til " + str(volume_of_box(1, 2, 3)) + " m3 i boksen")
```

```
Det er plass til 5 m3 i boksen
```

TRE FORMER FOR FEIL

- Syntaks
  - Programmet krasjer før det begynner å kjøre
  - Feilmelding gir visuell indikasjon på hva som er feil

```
IndentionError
SyntaxError
```

- Krasj
  - Programmet krasjer underveis i kjøring

```
AttributeError
IndexError
KeyError
NameError
TypeError
ZeroDivisionError
...
```

- Logiske feil
  - Programmet gir galt svar

ASSERT

- Krasj programmet med vilje når noe ikke er som det skal
- Tester koden, og beskytter mot logiske feil

```
assert(True)  # Gjør ingenting
assert(False) # Krasjer
```

- Vi bruker prinsippet om assert når vi retter kode automatisk (CodeGrade)
- Det er mulig å slå av assert for å optimisere kjøretid (men: ikke gjør det)
- Sjekk at assert er aktivt: legg inn `assert(False)` og se at det krasjer

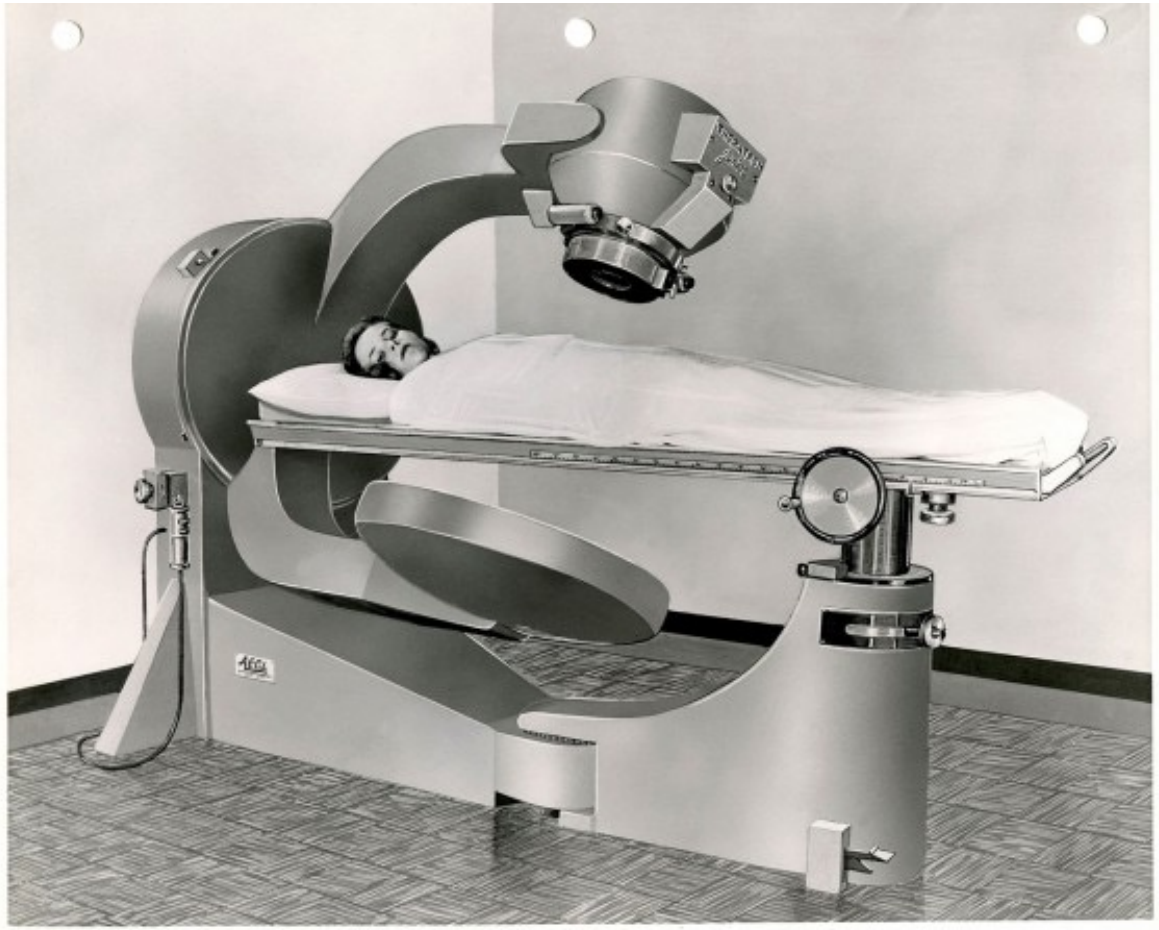
ASSERT

- Krasj programmet med vilje når noe ikke er som det skal
- Tester koden, og beskytter mot logiske feil

```
def volume_of_box(x, y, z):
    return (x * y + z)

assert(6 == volume_of_box(1, 2, 3))
print("Det er plass til " + str(volume_of_box(1, 2, 3)) + " m3 i boksen")
```

```
line 4, in <module>
    assert(6 == volume_of_box(1, 2, 3))
AssertionError
```

LOGISKE FEIL



Bilde: Canada Science and Technology Museum

Bilde: Jason Scott, CC-BY 2.0

UNIVERSITETET I BERGEN

UTTRYKK

INF100

HØST 2022

Torstein Strømme

UTTRYKK

Et *uttrykk* er

- Et "regnestykke" som evaluerer til en verdi
- En kombinasjon av verdier, variabler, funksjoner og operatorer

```
x = 2 + 2
x = (x + 1) * 2

a = min(x, 2 + 3)
if a == 5:
    print("Oj, a er 5")
```

UTTRYKK

Et *uttrykk* er

- Et "regnestykke" som evaluerer til en verdi
- En kombinasjon av verdier, variabler, funksjoner og operatorer

```
x = 2 + 2
x = (x + 1) * 2

a = min(x, 2 + 3)
if a == 5:
    print("Oj, a er 5")
```

UTTRYKK

Et *uttrykk* er

- Et "regnestykke" som evaluerer til en verdi
- En kombinasjon av verdier, variabler, funksjoner og operatorer

```
x = 2 + 2
x = (x + 1) * 2

a = min(x, 2 + 3)
if a == 5:
    print("Oj, a er 5")
```

UTTRYKK

Et *uttrykk* er

- Et "regnestykke" som evaluerer til en verdi
- En kombinasjon av verdier, variabler, funksjoner og operatorer

```
x = 2 + 2
x = (x + 1) * 2

a = min(x, 2 + 3)
if a == 5:
    print("Oj, a er 5")
```

UTTRYKK

Et *uttrykk* er

- Et "regnestykke" som evaluerer til en verdi
- En kombinasjon av verdier, variabler, funksjoner og operatorer

```
x = 2 + 2
x = (x + 1) * 2

a = min(x, 2 + 3)
if a == 5:
    print("Oj, a er 5")
```

UTTRYKK

Et *uttrykk* er

- Et "regnestykke" som evaluerer til en verdi
- En kombinasjon av verdier, variabler, funksjoner og operatorer

```
x = 2 + 2
x = (x + 1) * 2

a = min(x, 2 + 3)
if a == 5:
    print("Oj, a er 5")
```

UTTRYKK

Et *uttrykk* er

- Et "regnestykke" som evaluerer til en verdi
- En kombinasjon av verdier, variabler, funksjoner og operatorer

```
x = 2 + 2
x = (x + 1) * 2

a = min(x, 2 + 3)
if a == 5:
    print("Oj, a er 5")
```

UTTRYKK

Et *uttrykk* er

- en verdi, eller
- en variabel, eller
- et funksjonskall, eller
- to† uttrykk kombinert med en operator, eller
- et annet uttrykk omgitt av parenteser

```
"Howdy"
2.5
True
```

```
greeting
x
python_is_used
```

```
len(greeting)
volume_of_box(2, 5, x / 10)
circles_overlap(0, 0, 1, 1, 1, 1)
```

```
5 - 2
min(0, x) * 3
abs(x1 - x2) + abs(y1 - y2)
greeting + ", " + name + "!"
```

```
(2 + 2)
```

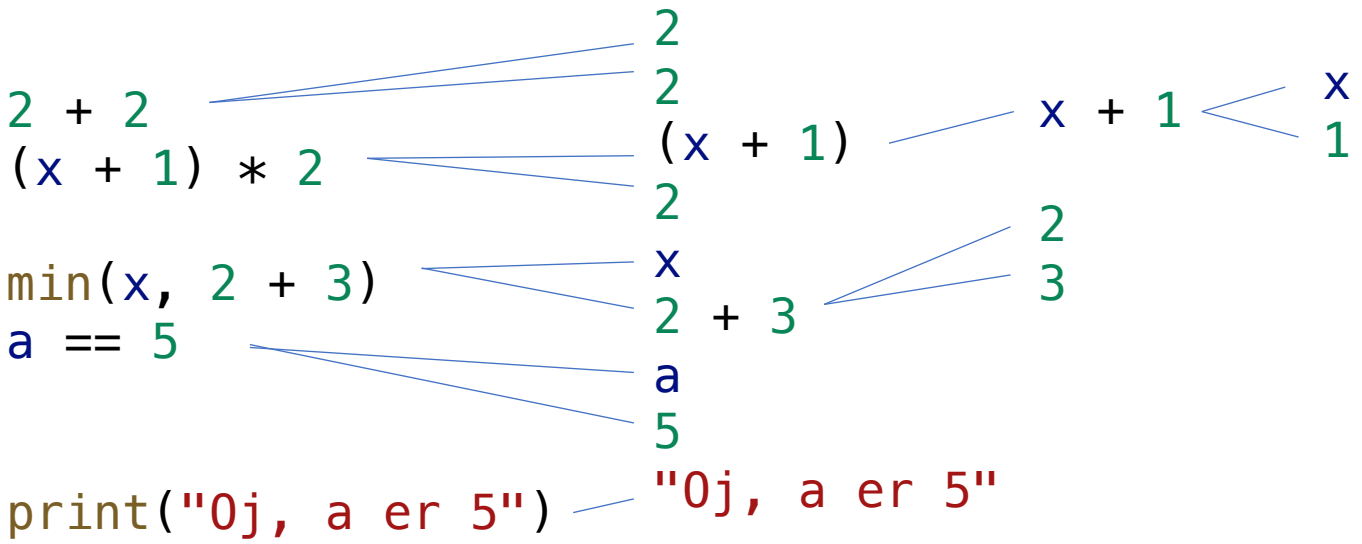
†Kan også være kun ett uttrykk (`not` …) eller tre uttrykk (… `if` … `else` …) avhengig av operator

UTTRYKK

Et *uttrykk* er

- Et "regnestykke" som evaluerer til en verdi
- En kombinasjon av verdier, variabler, funksjoner og operatorer

```
x = 2 + 2
x = (x + 1) * 2

a = min(x, 2 + 3)
if a == 5:
    print("Oj, a er 5")
```



“99% of people fail to solve this problem”

$$6/2(1+2)$$

www.menti.com
3564 3026

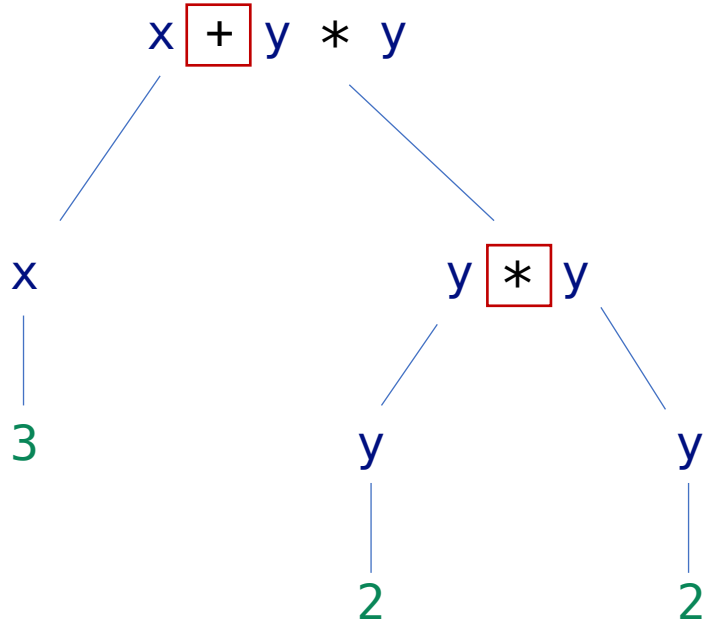
EVALUERING AV UTTRYKK

- Hvis et uttrykk er en verdi, er uttrykket ferdig evaluert.
- Hvis uttrykket er en variabel, evalueres uttrykket til den verdien variabelen referer til.
- Hvis uttrykket er inne i en parentes, fjernes parantesene og uttrykket mellom parentesene evalueres.
- Ellers:
  - Velg den operatoren med lavest presedens lengst til høyre★, og del uttrykket i to:
    - Evaluer venstresiden av uttrykket
    - Evaluer høyresiden av uttrykket
    - Kombiner verdiene med operatoren

★ unntak for **-operatøren, der velges den lengst til venstre.

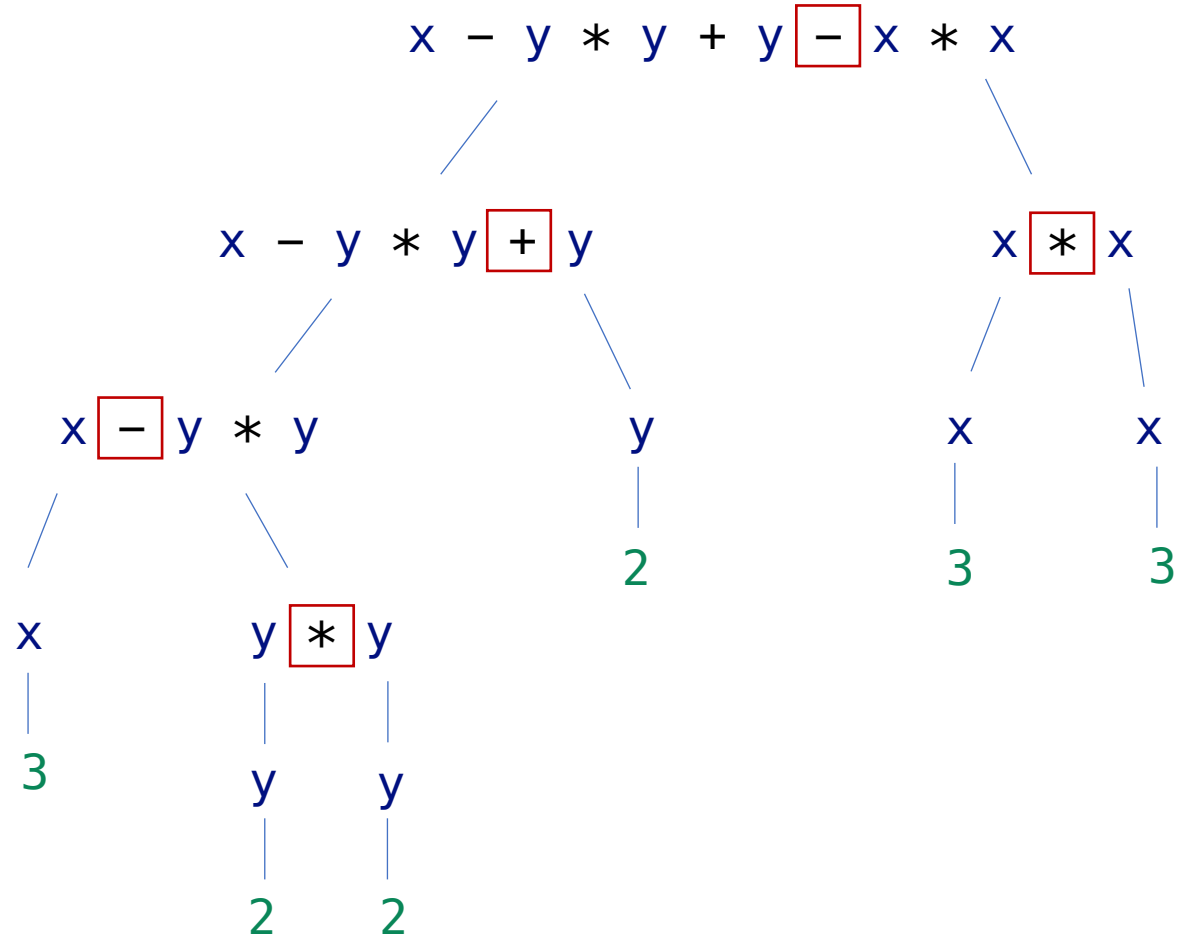
EKSEMPLER

x = 3
y = 2



EKSEMPLER

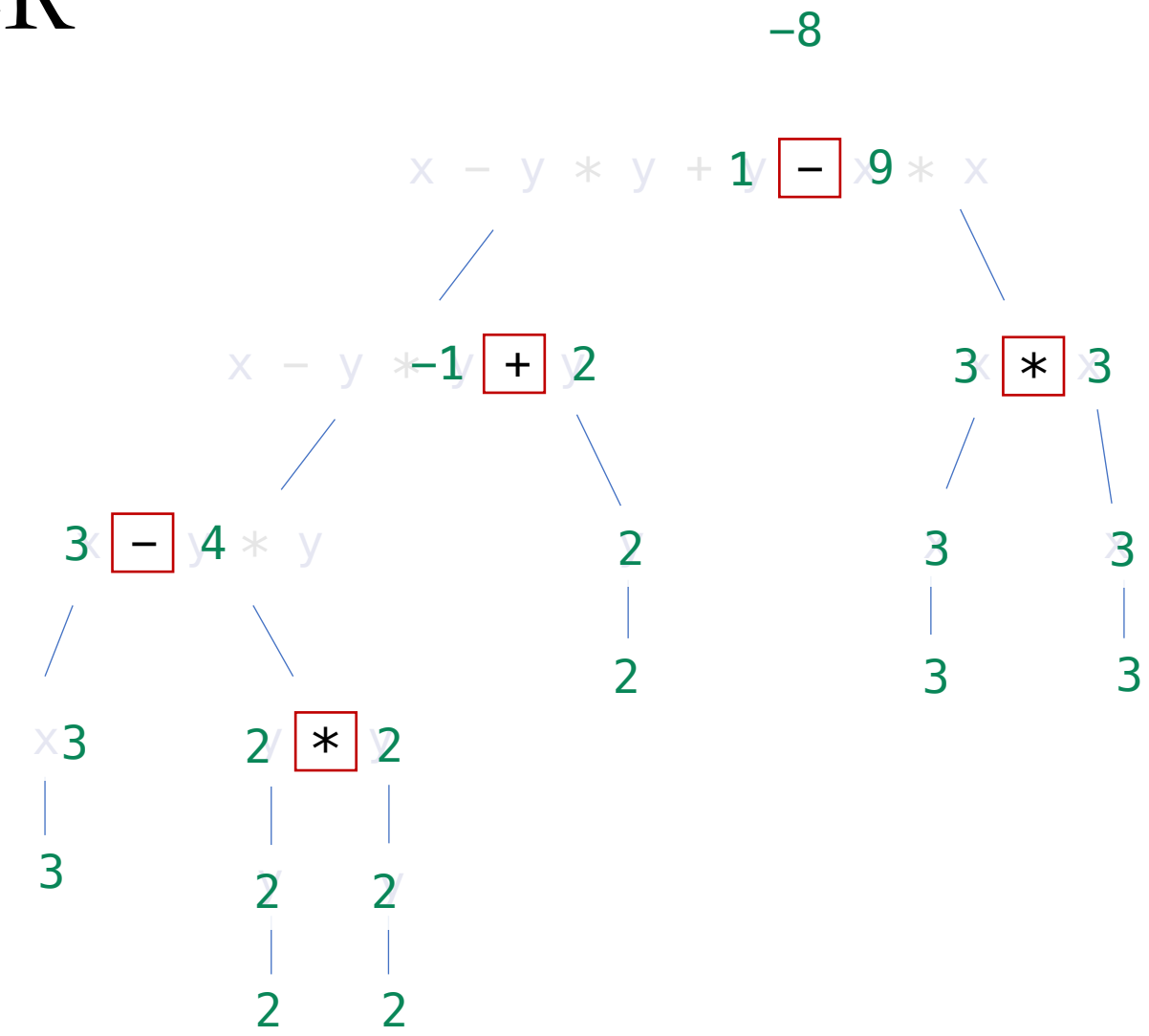
x = 3
y = 2



((x - (y * y)) + y) - (x * x)

EKSEMPLER

x = 3
y = 2



((x − (y * y)) + y) − (x * x)

EVALUERING AV UTTRYKK

- Tommelfingerregel:
  - Operasjoner med høy presedens utføres først
  - Operasjoner med lik presedens utføres fra venstre mot høyre (unntak: **)
  - Evaluering av funksjoner og variabler skjer fra venstre mot høyre (unntak: if else)

- Takeaway:
  - Benytt parenteser!

KORTSLUTNINGSEVALUERING

```
def true1():
    print("true1", end=" ")
    return True

def true2():
    print("true2", end=" ")
    return True

def false1():
    print("false1", end=" ")
    return False

def false2():
    print("false2", end=" ")
    return False
```

```
if true1():
    print("johoo")
```

```
if false1() or true1():
    print("johoo")
```

```
if true1() or true2():
    print("johoo")
```

KORTSLUTNINGSEVALUERING

- Dersom venstreside av or –uttrykk er True, evlaueres ikke høyre side
- Dersom venstreside av and—uttrykk er False, evalueres ikke høyre side