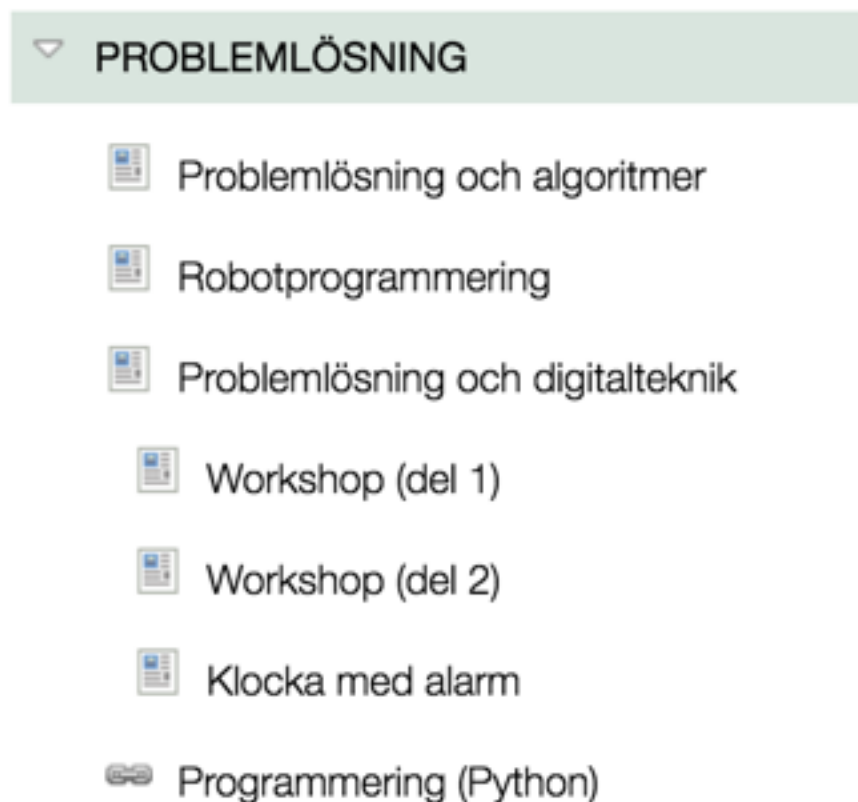


1 skärm = ett par ur basgruppen

Vilket par du tillhör framgår på kursens sida i **Studentportalen** ➤ **Problemlösning** ➤ **Programmering (Python)** ➤ **Del 2** ➤ **Syfte och översikt** ➤ **Arbete och redovisning i par om två studenter.**



Introduktion till informationsteknologi (1DT051)








Programmering (2) - workshop

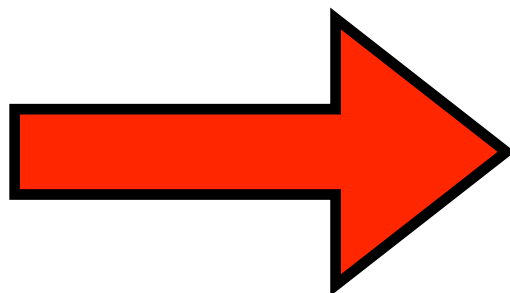
Förberedelser

(1)

Från kursens sida i Studentportalen, öppna hemsidan om problemlösning och programmering.

▼ PROBLEMLÖSNING

-  Problemlösning och algoritmer
-  Robotprogrammering
-  Problemlösning och digitalteknik
-  Workshop (del 1)
-  Workshop (del 2)
-  Klocka med alarm
-  Programmering (Python)



Introduktion till informationsteknologi

1DT051/2016

Problemlösning och programmering

Start

Syfte och översikt

Syfte

Förkunskaper

Förberedelser

Innehåll

Arbete i par om två studenter

Workshops

Problemlösning och programmering

¶

På kursen **Introduktion till informationsteknologi (1DT051)** ingår momentet problemlösning och programmering. All information om detta moment hittar du på dessa sidor.

Syfte

¶

Syftet med detta moment är att se hur metoden **söndra och härska** kan användas för att lösa problem med hjälp av programmering. Du får prova på och lära sig grunderna inom programmering och då särskilt i det **imperativa** programmeringsspråket **Python**.

Förberedelser

(2)

Navigera till Del 2 ➤ Workshop ➤ Python tutor

Del 2

Syfte och översikt

Workshop

Python tutor

Konfigurerings av texeditor

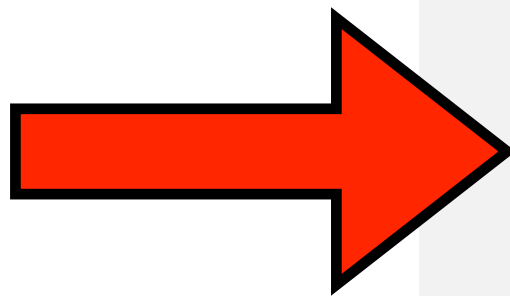
Slides (pdf)

Uppgifter

Frivillig fördjupning

Handledning

Redovisning



Python tutor

Under workshopen kommer vi kika på följande exempel i Python Tutor:

- Slingan **while**
- Slingan **for-in**
- Slingan **for-in som specialfall av while**
- Funktionen **max()**
- Funktionen **length()**
- **Räckvid (scope) och sidoeffekter**
- Exempel på **högre ordningens funktioner**
- Funktionen **is_palindrome()**

Repetition

En kort sammanfattning av det vi lärt oss om Python och programmering under del 1.

Aritmetiska operationer i Python

Addition	A + B
Subtraktion	A - B
Multiplikation	A * B
Division	A / B
Modulo	A % B
Exponent	A ** B

Logiska operationer i Python

Lika med	A == B
Skilt från	A != B
Mindre än	A < B
Större än	A > B
Mindre eller lika med	A <= B
Större eller lika med	A >= B
Logiskt och	A and B
Logiskt eller	A or B
Logiskt icke	not A

Datatyp

Vad menas med begreppet datatyp?

I programspråk är en datatyp ett *attribut* för data som berättar för datorn (och programmeraren) *vilken sorts information* datat bär på.

Eftersom all information i datorn, även text och bilder, internt hanteras som tal är datatyper ett sätt att se *skillnad* på vad talen *representerar*.

Datatyper i Python

Vi har tidigare bekantat oss med följande datatyper i Python.

Datatyp			
Svenska	Python	Exempel	
Heltal	<code>int</code>	13	127
Decimaltal	<code>float</code>	13.0	0.333333
Sanningsvärde	<code>bool</code>	True	False
Sträng	<code>str</code>	"Hej Bosse!"	'A@!&_?'

Kontrollstrukturer i Python

Vi har tidigare bekantat oss med kontrollstrukturen **if-elif-else**.

```
x = 10
```

Notera kolon (:) och indrag (tab).

```
if x > 100:  
    print "x is huge!"  
elif x > 50:  
    print "x is large!"  
elif x > 40:  
    print "x is medium!"  
else:  
    print "x is small!"
```

`magic_sum(a, b, c)`

Skriv en funktion med namn `magic_sum()` som tar tre tal som argument och returnerar summan av de fyrdubbla värdena av de två största talen.

- Förstå problemet
- Vad utgör problemets kärna?
 - Jo - Hur hitta de **två största talen av tre tal!**

`magic_sum(a, b, c)`

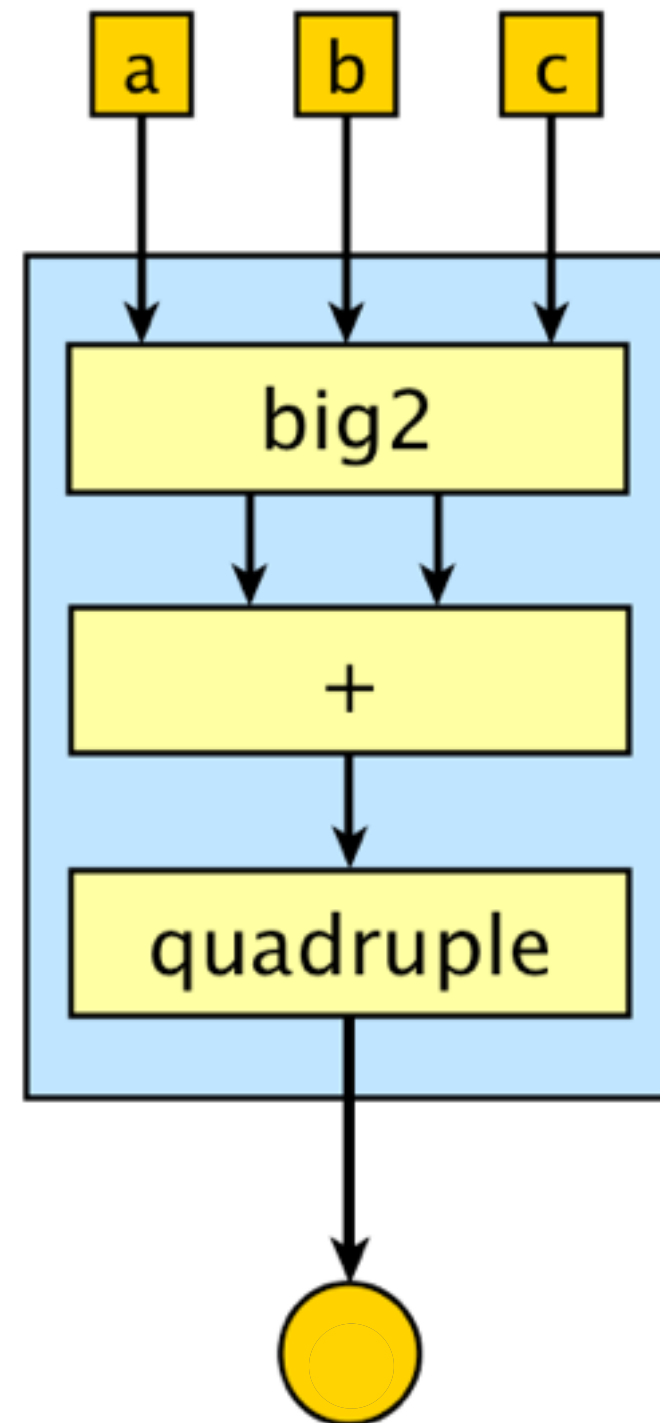
Skriv en funktion med namn `magic_sum()` som tar tre tal som argument och returnerar summan av de fyrdubbla värdena av de två största talen.

```
def magic_sum(a, b, c):  
    m1 = max3(a, b, c)  
    if a == m1:  
        m2 = max2(b, c)  
    elif b == m1:  
        m2 = max2(a, c)  
    else:  
        m2 = max2(a, b)  
    return quadruple(m1+m2)
```

`magic_sum(a, b, c)`

(1)

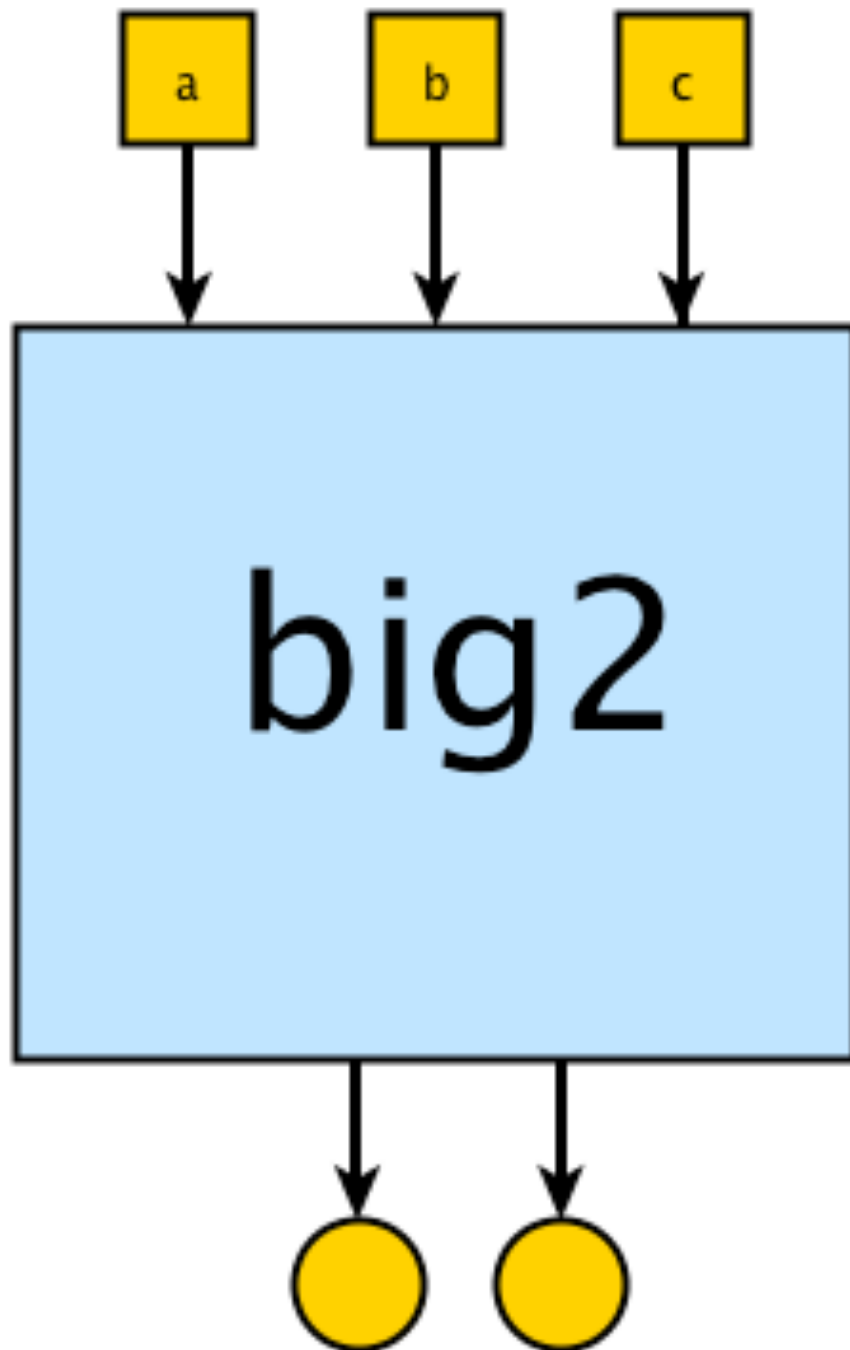
Grafiskt representation av funktionen `magic_sum`.



big2(a, b, c)

(1)

Hur kan vi skriva en funktion som returnerar de två största talen av tre tal?



Söndra och härska!

Hur returnera två värden från en funktion?

Tupel

Inom matematiken är en **tupel** en ändlig objektsekvens, vars komponenter har var för sig en bestämd typ.

En tupel bestående av n objekt kallas **n -tupel**.

Till exempel kan 4-tupeln vars komponenter är PERSON, ÅR, MÅNAD, DAG användas för att registrera att en person är född på en viss dag i en viss månad i ett visst år.

Tupler i Python

I Python kan tupler användas för att samla ihop sådant som hör ihop på något sätt.

- En 2-tupel (vanligen kallad endast tupel) består av två element inom parenteser separerade med ett kommatecken.
- En 3-tupel består på liknande sätt av 3 element inom parenteser separerade med kommatecken.

```
>>> ("Bosse", 666)
('Bosse', 666)
>>> (0xf, "apa", 0.75)
(15, 'apa', 0.75)
>>>
```

Här skapar vi en tupel med elementen "Bosse" (str) och 666 (heltal).

Här skapar vi en 3-tupel med elementen 15 (int), "apa" (str) och 0.75 (float).

Tupel-indexering (framlänges)

Det första elementet har index 0, det andra elementet index 1 osv. Med hjälp av index inom hak-parenteser [och] fås värdet på ett element i en tuple.

```
>>> ("Bosse", 666)[0]
'Bosse'
>>> ("Bosse", 666)[1]
666
```

Index kan även användas när en tuple lagrats i en variabel.

```
>>> t = ("Bosse", 666)
>>> t[0]
'Bosse'
>>> t[1]
666
```

Tupel-indexering (baklänges)

Tupler kan även indexeras bakifrån, då har det sista elementet index -1, det näst sista index -2 osv.

```
>>> ("Bosse", 666)[-1]
666
>>> ("Bosse", 666)[-2]
'Bosse'
```

Index kan även användas när en tuple lagrats i en variabel.

```
>>> t = ("Bosse", 666)
>>> t[-1]
666
>>> t[-2]
'Bosse'
```

Tupel-matchning

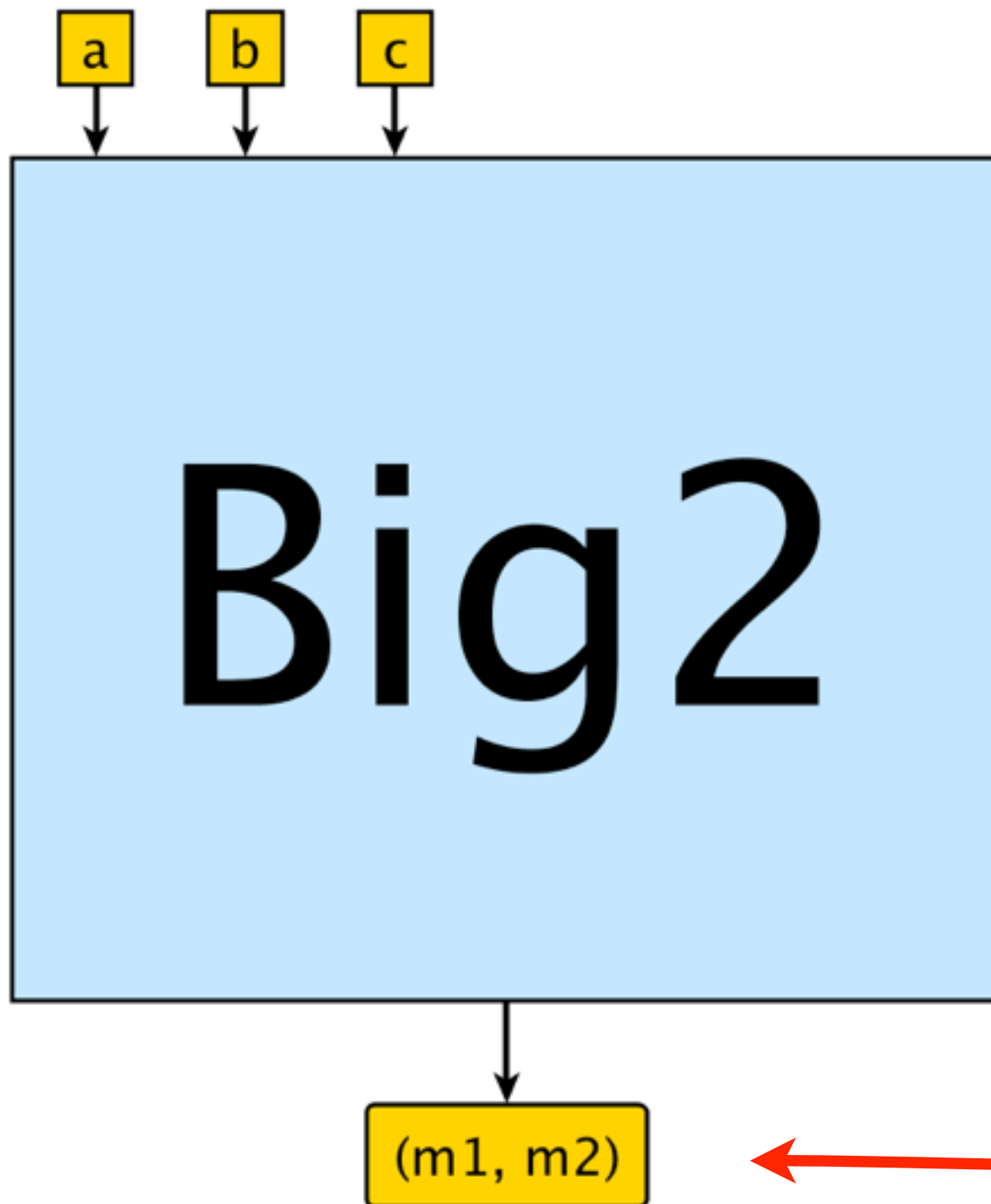
Med operatorn för tilldelning **=** kan en tupel plockas i sär och de olika element lagras i separata variabler.

```
>>> x = ("Bosse", 666) # Skapa en tupel.
>>> (name, number) = x # Plocka i sär ...
>>> name               # Variabeln name ...
'Bosse'                # ... innehåller "Bosse".
>>> number              # Variabeln number ...
666                    # ... innehåller 666.
>>>
```

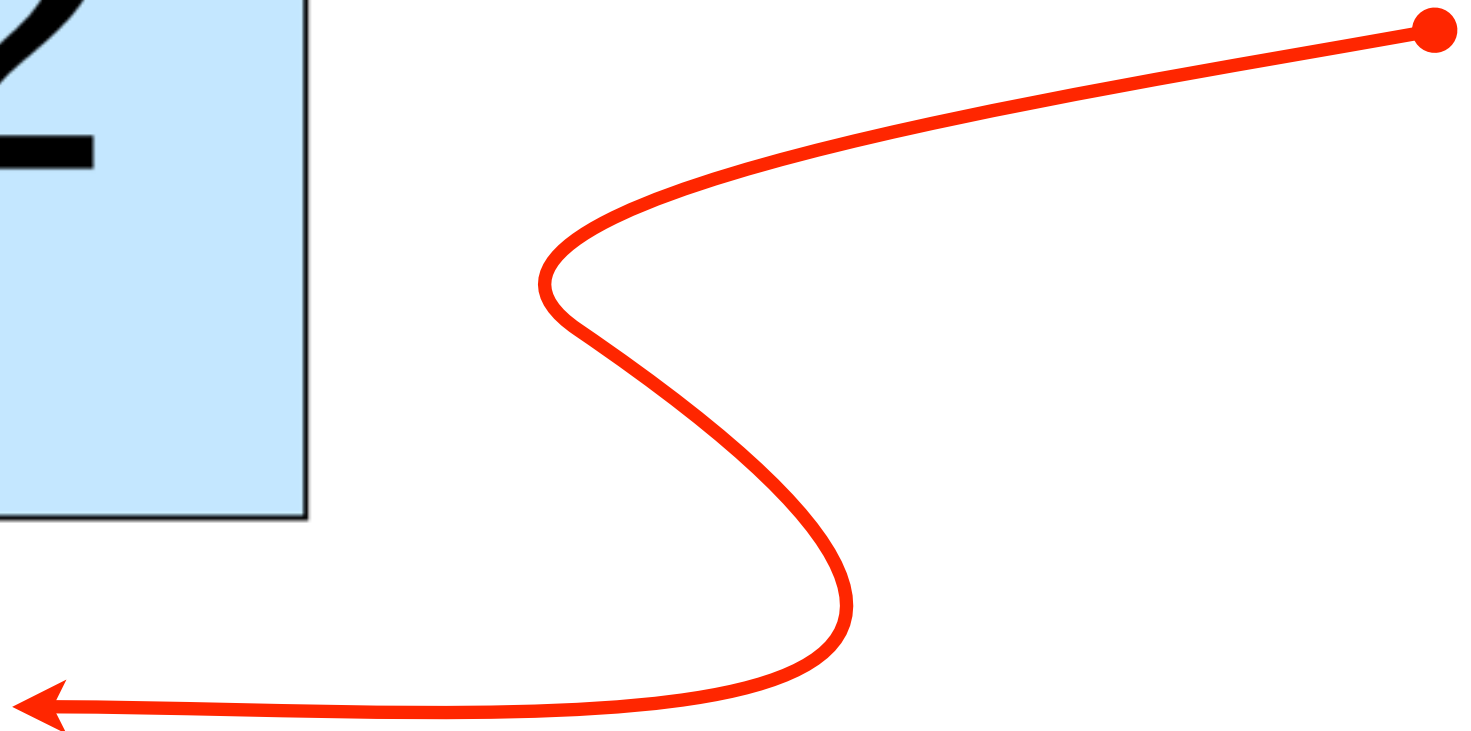
big2(a, b, c)

(2)

Hur kan vi skriva en funktion som returnerar de två största talen av tre tal?

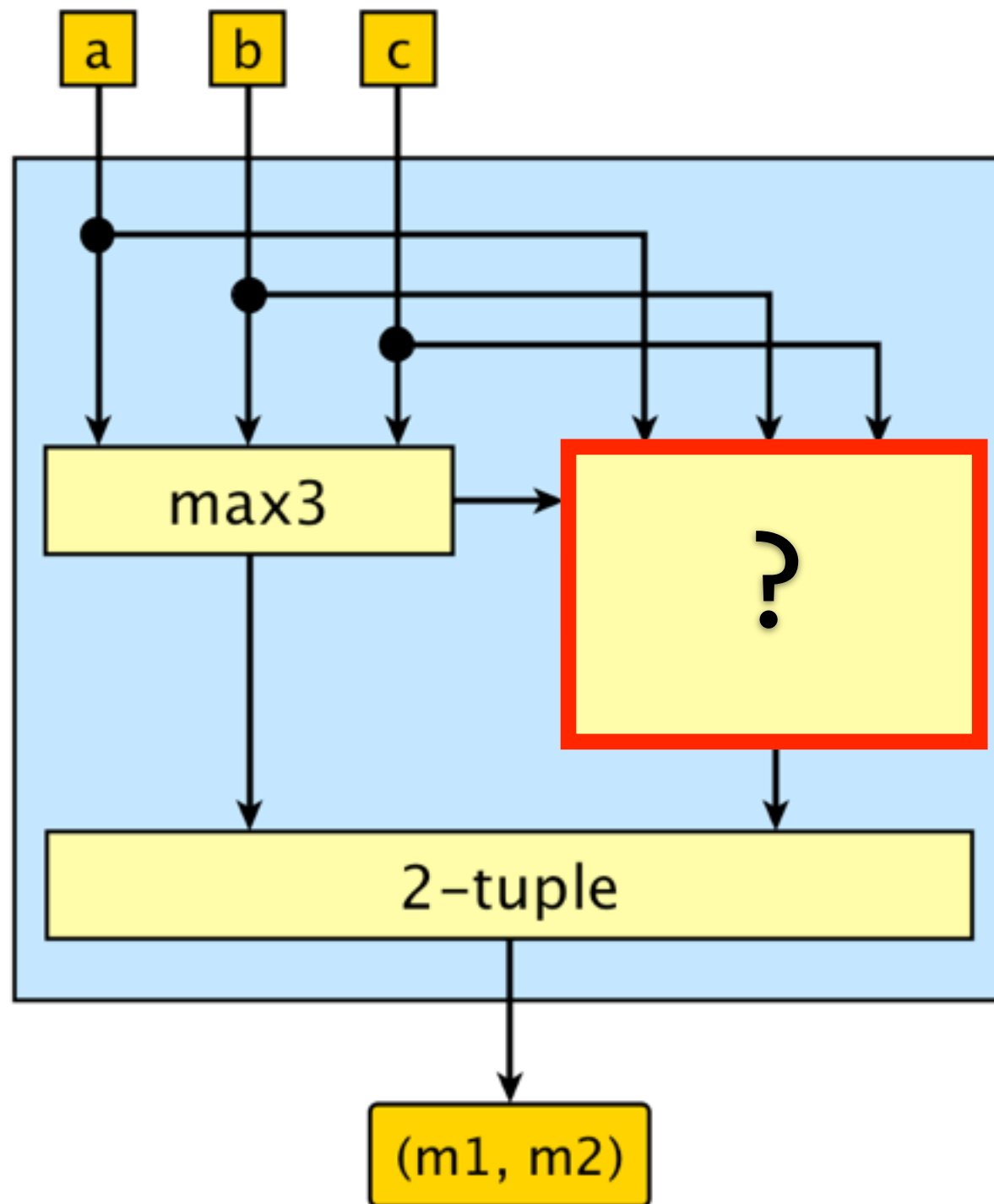


returnera en 2-tupel



(3)

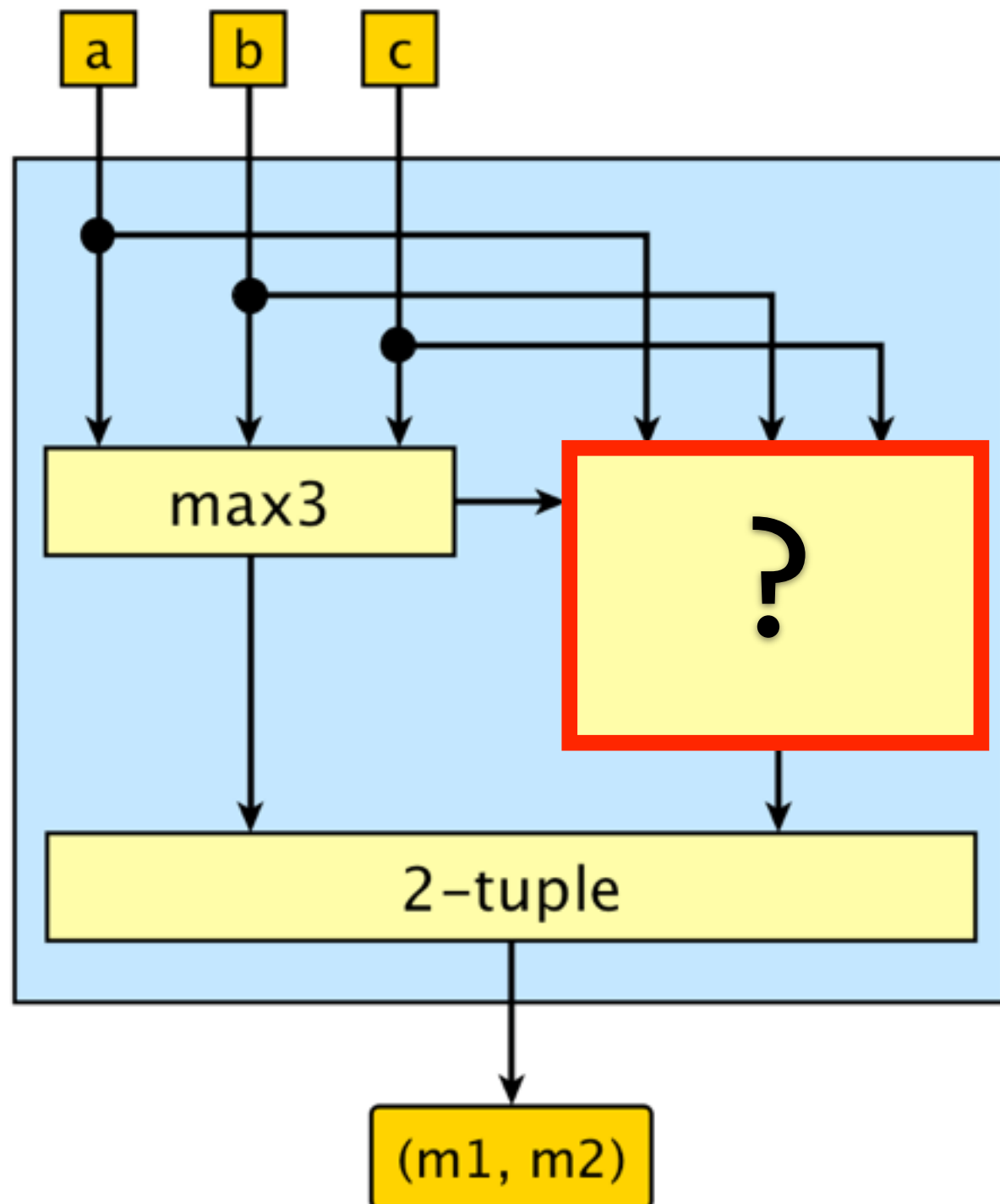
Hur kan vi skriva en funktion som returnerar de två största talen av tre tal?



big2(a, b, c)

(4)

Hur kan vi skriva en funktion som returnerar de två största talen av tre tal?



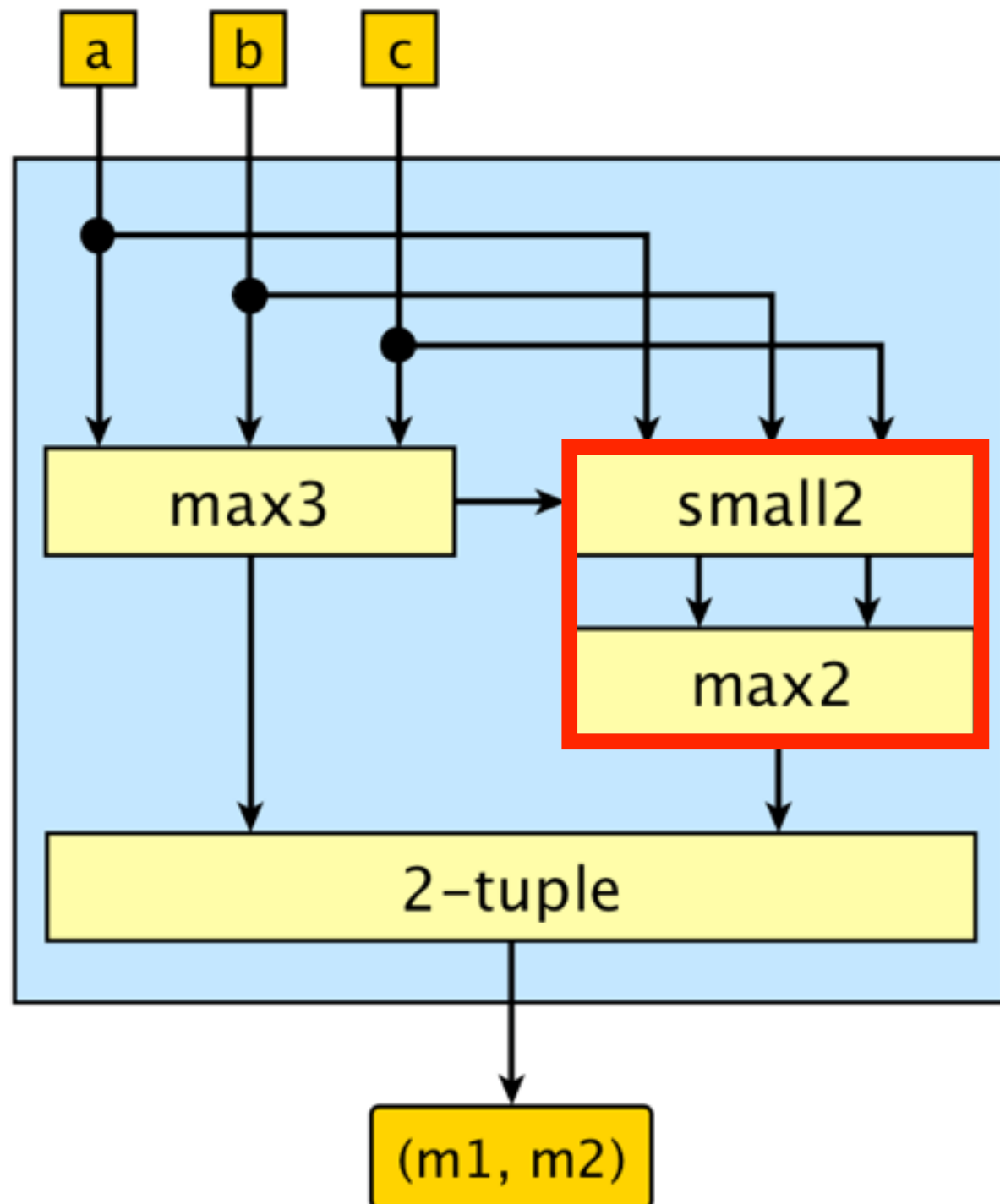
Använd if-elif-else

```
def big2(a, b, c):  
    m = max3(a, b, c)  
    if a == m:  
        return (a, max2(b, c))  
    elif b == m:  
        return (b, max2(a, c))  
    else:  
        return (c, max2(a, b))
```

big2(a, b, c)

(4)

Hur kan vi skriva en funktion som returnerar de två största talen av tre tal?

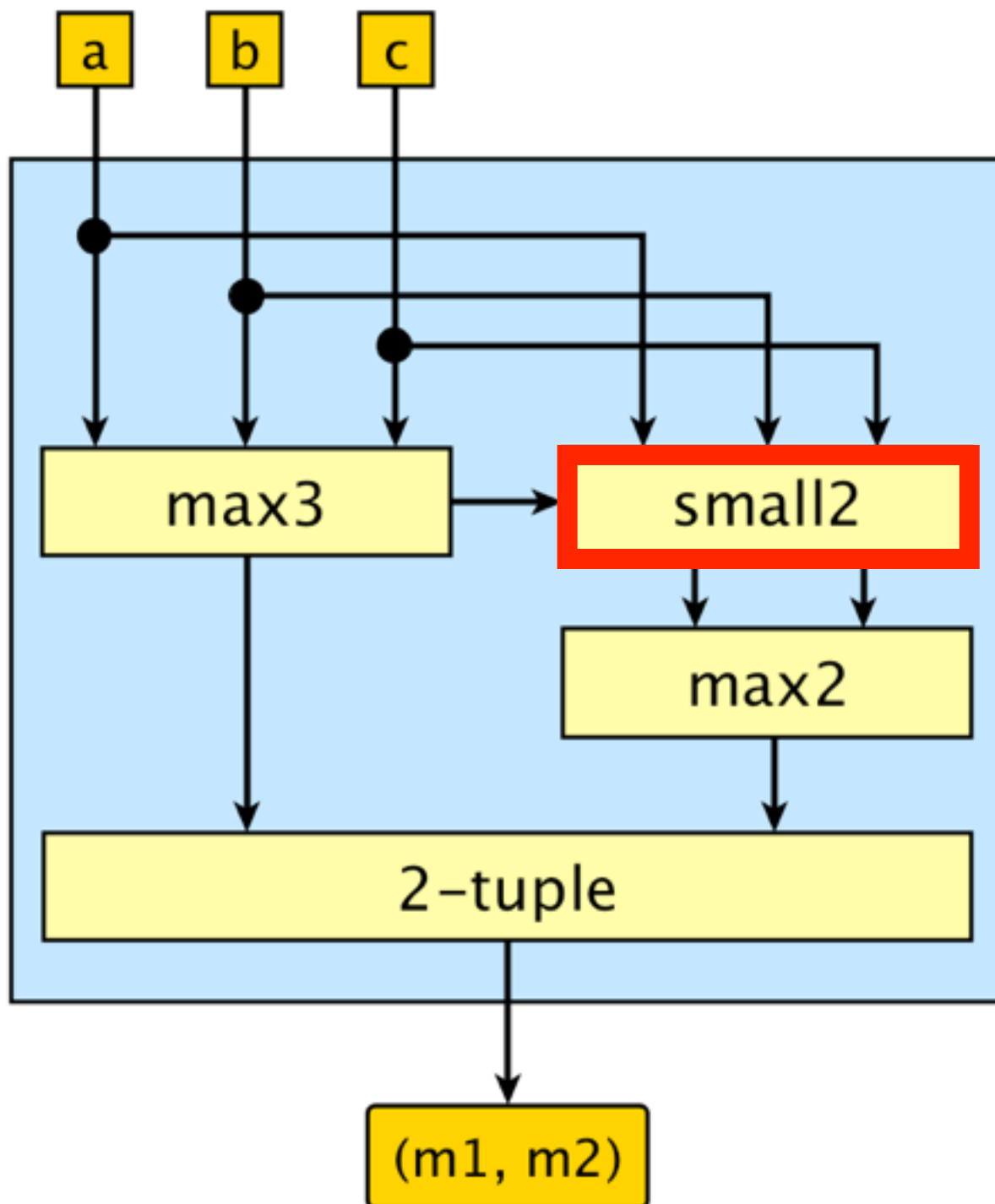


Använd if-elif-else

```
def big2(a, b, c):
    m = max3(a, b, c)
    if a == m:
        return (a, max2(b, c))
    elif b == m:
        return (b, max2(a, c))
    else:
        return (c, max2(a, b))
```


small2(m, a, b, c)

Givet att vi vet det största värdet, hur kan vi skriva en funktion som returnerar de två minsta talen av tre tal?

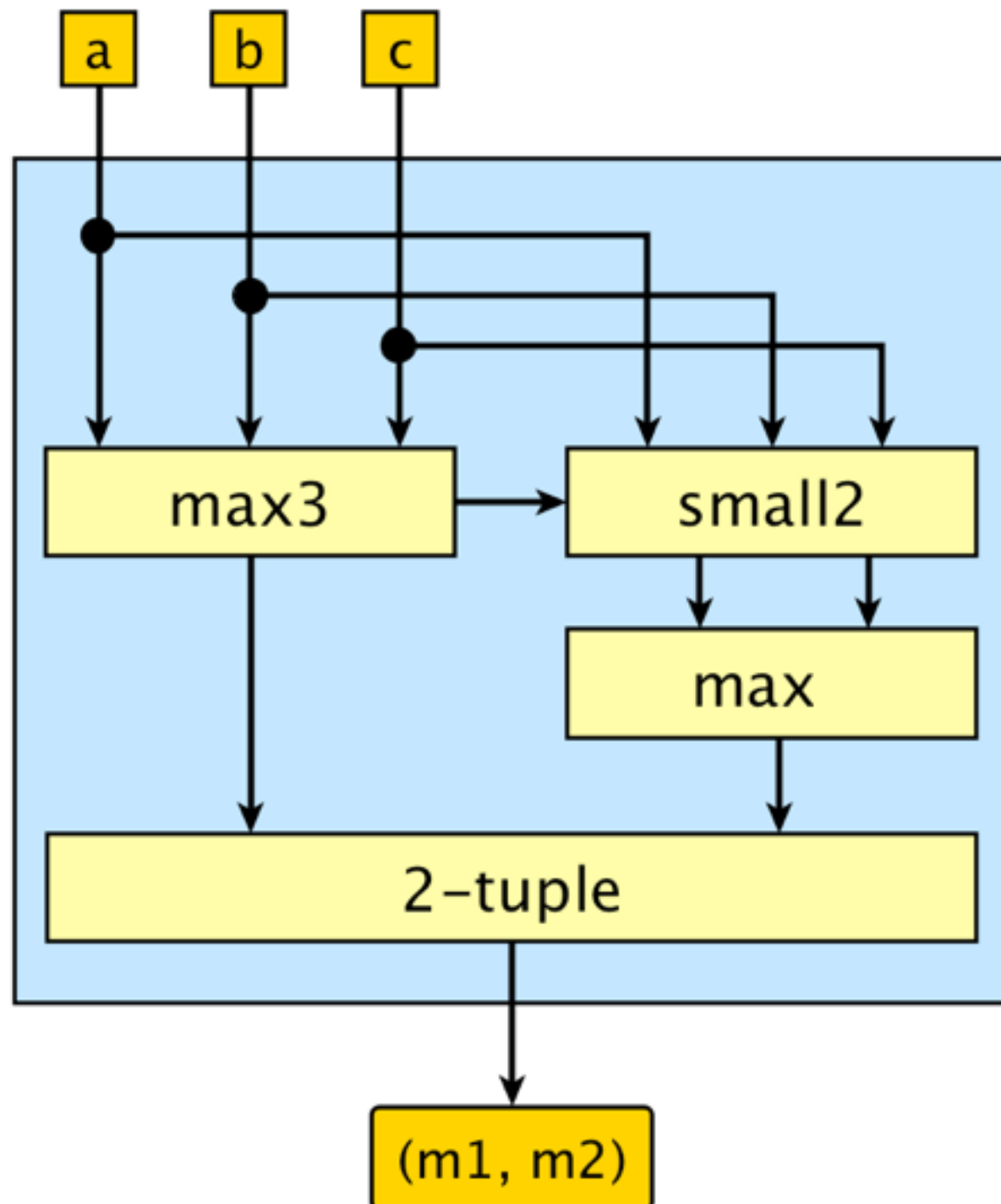


```
def small2(max, a, b, c):  
    if a == max:  
        return (b, c)  
    elif b == max:  
        return (a, c)  
    else:  
        return (a, b)
```

big2(a, b, c)

(5)

Hur kan vi skriva en funktion som returnerar de två största talen av tre tal?



Använd **small2** och **max**

```
def big2(a, b, c):  
    m = max3(a, b, c)  
    return (m, max(small2(m, a, b, c)))
```

max är en inbyggd funktion i Python som returnerar det största talet i en lista eller tupel.

big2(a, b, c)

(6)

Hur kan vi skriva en funktion som returnerar de två största talen av tre tal?

```
>>> big2(77, 5, 127)
```

```
(127, 77)
```

```
>>> (m1, m2) = big2(1, 333, -99)
```

```
>>> m1
```

```
333
```

```
>>> m2
```

```
1
```

```
>>>
```

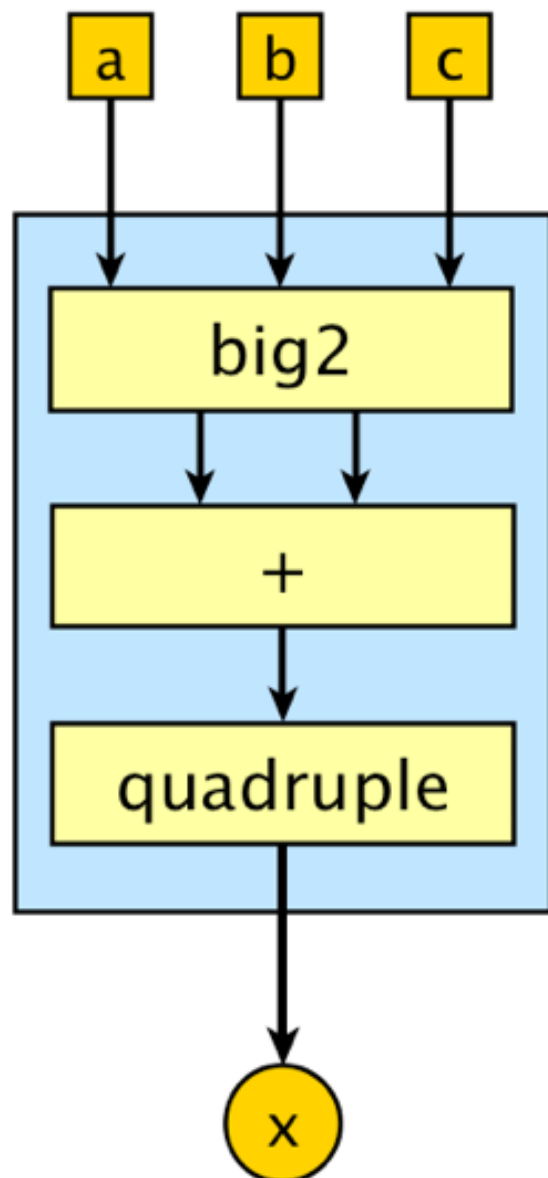
`magic_sum(a, b, c)`

Skriv en funktion med namn `magic_sum()` som tar tre tal som argument och returnerar summan av de fyrdubbla värdena av de två största talen.

```
def magic_sum(a, b, c):  
    m1 = max3(a, b, c)  
    if a == m1:  
        m2 = max2(b, c)  
    elif b == m1:  
        m2 = max2(a, c)  
    else:  
        m2 = max2(a, b)  
    return quadruple(m1+m1)
```

`magic_sum_v2(a, b, c)`

Skriv en funktion med namn `magic_sum()` som tar tre tal som argument och returnerar summan av de fyrdubbla värdena av de två största talen.



```
def magic_sum_v2(a, b, c):  
    (x, y) = big2(a, b, c)  
    return quadruple(x + y)
```

Söndra och härska: Genom att bryta ner problemet i mindre delar och lösa dessa var för sig kan ett till en början knivigt problem slutligen lösas på ett enkelt sätt.

Slingor (loopar)

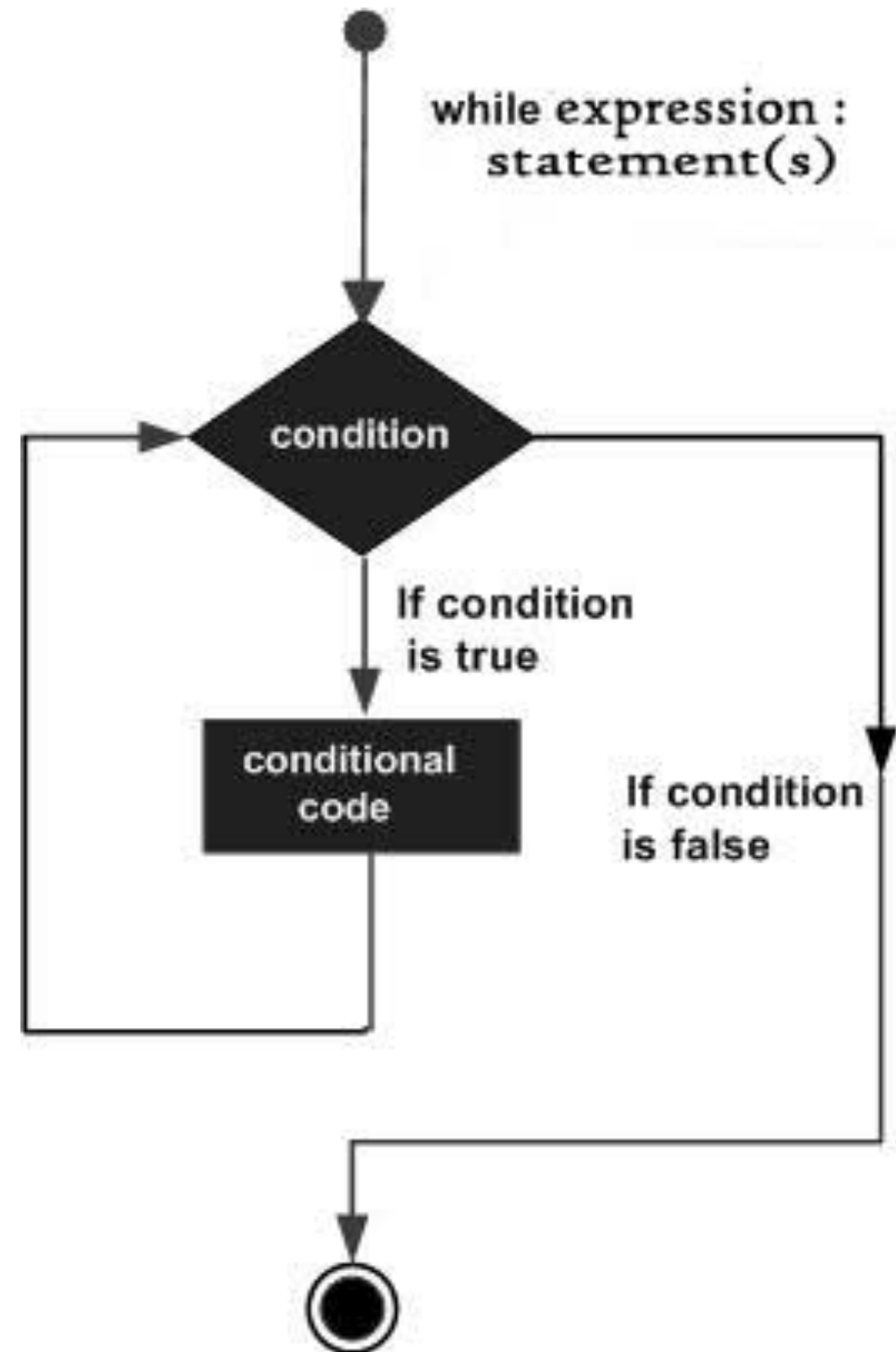
Vad menas med begreppet slinga?

En slinga eller programslinga (engelska loop) är en konstruktion inom **imperativa programmeringsspråk** för att åstadkomma en **iteration**, så att en serie **satser upprepas** flera gånger.

Antalet upprepningar kan vara förbestämt, men kan även bestämmas dynamiskt. Slingor används ofta då samma uppgift skall utföras för en mängd element, som att skriva ut alla namn som finns i en lista.

While

Ett sätt att åstadkomma en slinga i Python är med hjälp av **kontrollstrukturen** **while**.



While

Ett sätt att åstadkomma en slinga i Python är med hjälp av kontrollstrukturen **while**.

villkor

kolon

```
>>> x = 0
```

```
>>> while x < 4:
```

```
... |
```

```
print x
```

```
...
```

```
x = x + 1
```

```
...
```

Det som skall utföras om villkoret är sant måste skrivas efter ett indrag (tryck tab en gång).

While

Ett sätt att åstadkomma en slinga i Python är med hjälp av kontrollstrukturen **while**.

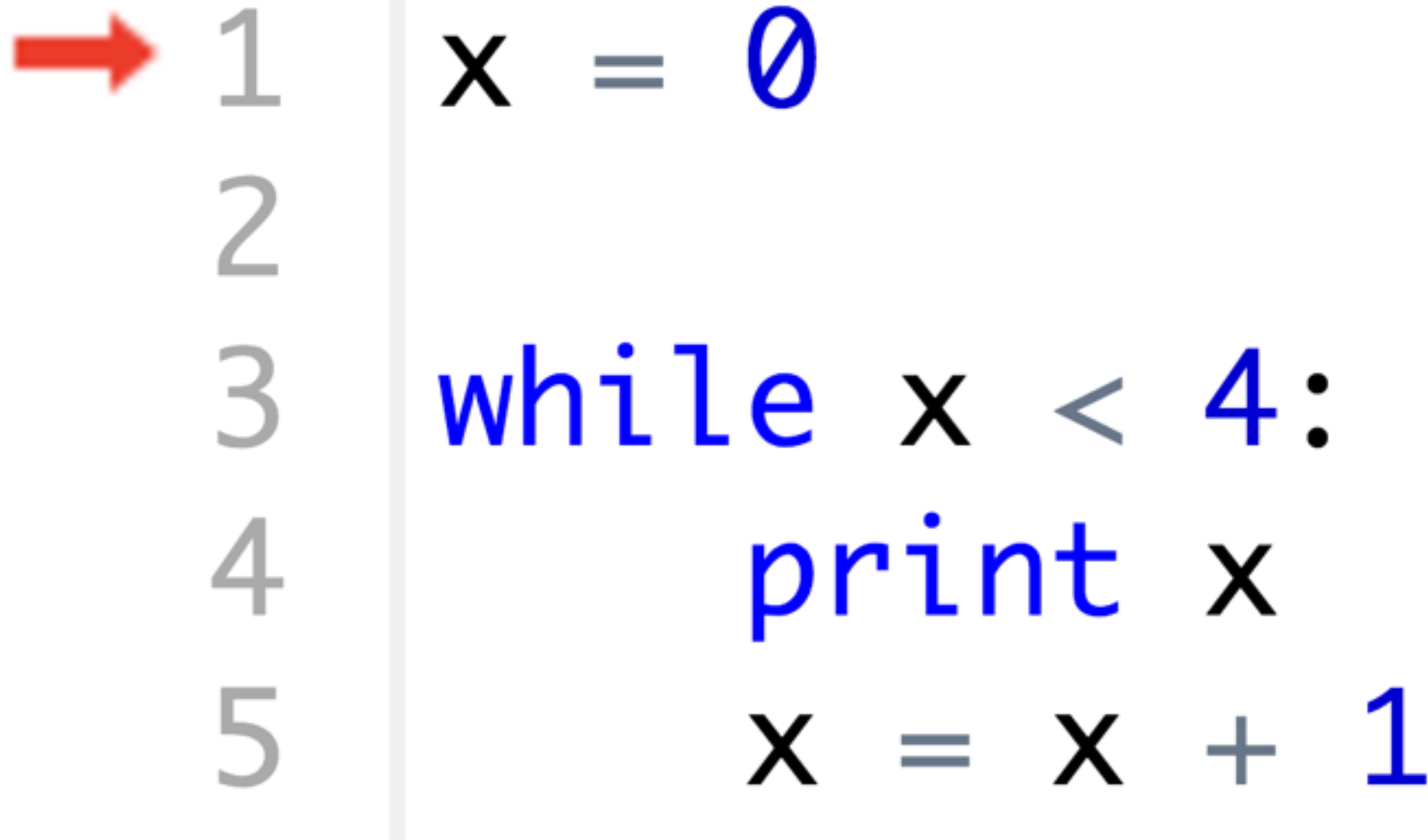
I detta fall skrivs heltalen 0 till 4 ut ett och ett på varsin rad.

```
>>> x = 0
>>> while x < 5:
...     print x
...     x = x + 1
... 
```

```
0
1
2
3
4
>>>
```

Python Tutor

Experimentera med slingan **while** i Python tutor.



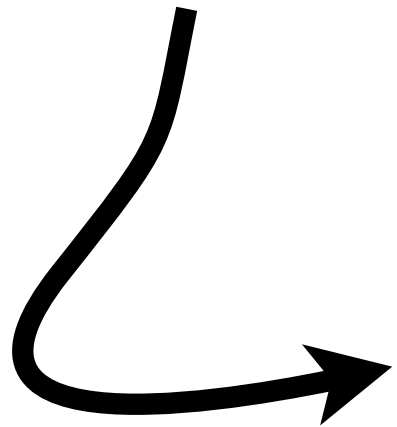
The image shows a Python Tutor interface. On the left, there is a vertical list of line numbers 1 through 5. A red arrow points to line 1. To the right of the line numbers, the code is displayed. Line 1: `x = 0`. Line 3: `while x < 4:`. Line 4: `print x`. Line 5: `x = x + 1`. The code is color-coded: `while`, `print`, and the final `1` are blue, while the other tokens are black.

```
1 x = 0
2
3 while x < 4:
4     print x
5     x = x + 1
```

Länk till **Python Tutor** med detta exempel finns på sidan **Del 2 - workshop** ➤ **Python Tutor**.

Starta en terminal

Din prompt kommer se annorlunda ut.



```
linux>
```

Skriv python vid Linux-prompten och tryck enter.



```
linux> python
```

Listor

Listor är praktiska och används flitigt i Python.

En lista kännetecknas av att den omges av hakparenteser `[` och `]`. Listans element separeras av kommatecken.

```
>>> a = []           # En tom lista.  
>>> b = [1, 2, 77]   # En lista med tre element.
```

Det går utmärkt att blanda olika typer av värden i en och samma lista, till exempel heltal, strängar och andra listor.

```
>>> c = [8, 9, "apa", ["en", "lista", "i", "en", "lista"], 127]
```

Vilken datatyp har en lista?

Med hjälp av den inbyggda funktionen `type()` kan vi ta reda på vilken datatyp något har i Python.

```
>>> z = [1, "apa", [55, 20], 77]
>>> type(z)
<type 'list'>
>>>
```

```
>>> type("bosse")
<type 'str'>

>>> type(5+3)
<type 'int'>
```

```
>>> type(1.0/3)
<type 'float'>

>>> type(5 > 8)
<type 'bool'>
```

Datatyper i Python

Datatyp				
Svenska	Python	Exempel		
Heltal	<code>int</code>	13	-666	127
Decimaltal	<code>float</code>	13.0	5.125	-0.333333
Sanningsvärde	<code>bool</code>	True	False	
Tupel	<code>tuple</code>	(127, 'hej')	(1, "apa", 4.25)	
Sträng	<code>str</code>	"Bosse"	'Nisse'	'#@!&_?'
Lista	<code>list</code>	[]	[1,2,3]	[1, [2,3], 4]

Längden på en lista

Den inbyggda funktion **len()** används för att ta reda på hur många element en lista innehåller.

```
>>> a = [0, 1, 2, 3, 4]
```

```
>>> len(a)
```

```
5
```

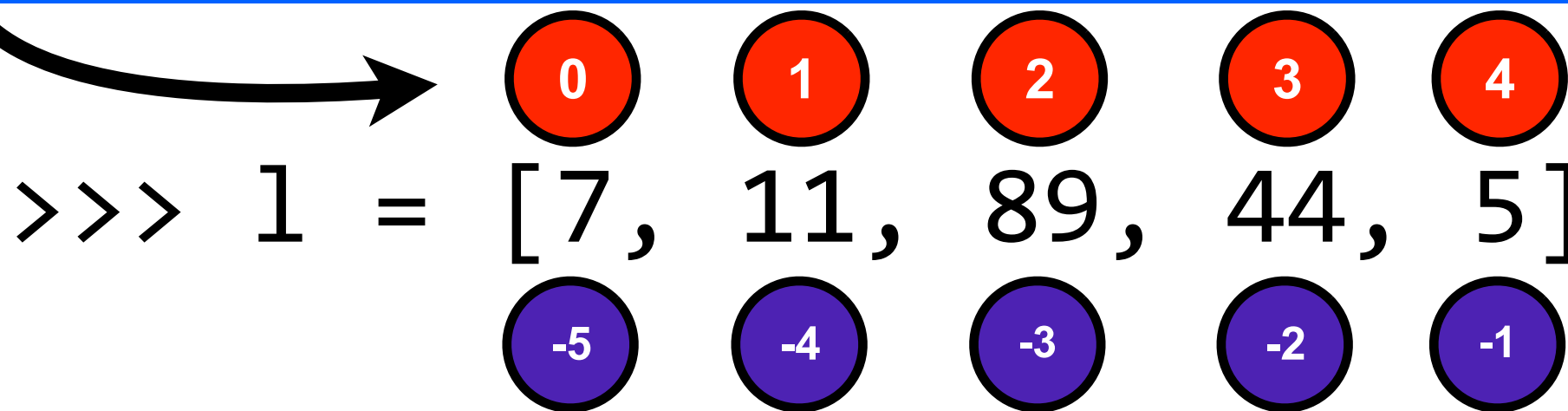
```
>>> len([])
```

```
0
```

Index

Elementen i en lista indexeras så att det första elementet har index 0, det andra elementet index 1 osv. Med hjälp av negativa index kan listan indexeras baklänges.

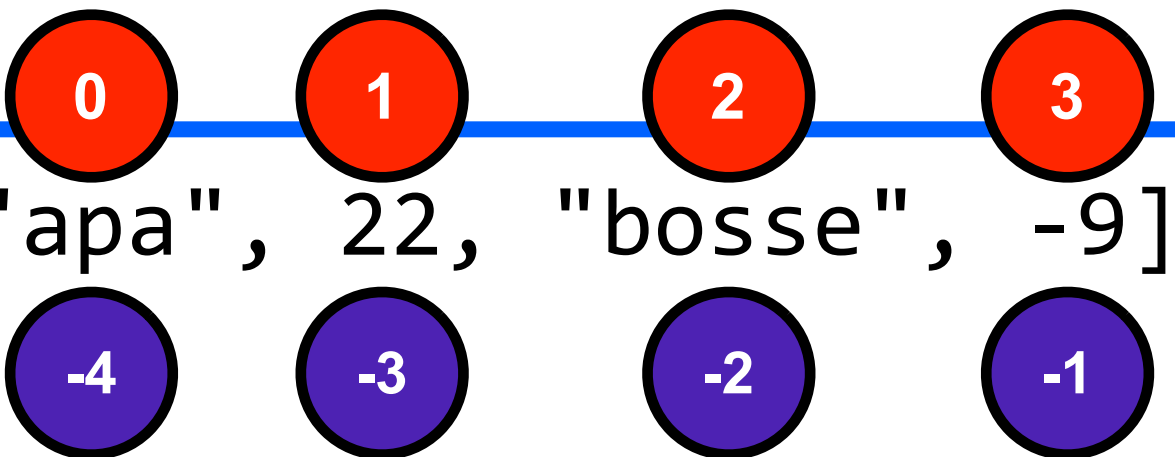
index framlänges



index baklänges

Värden på enskilda element i en lista

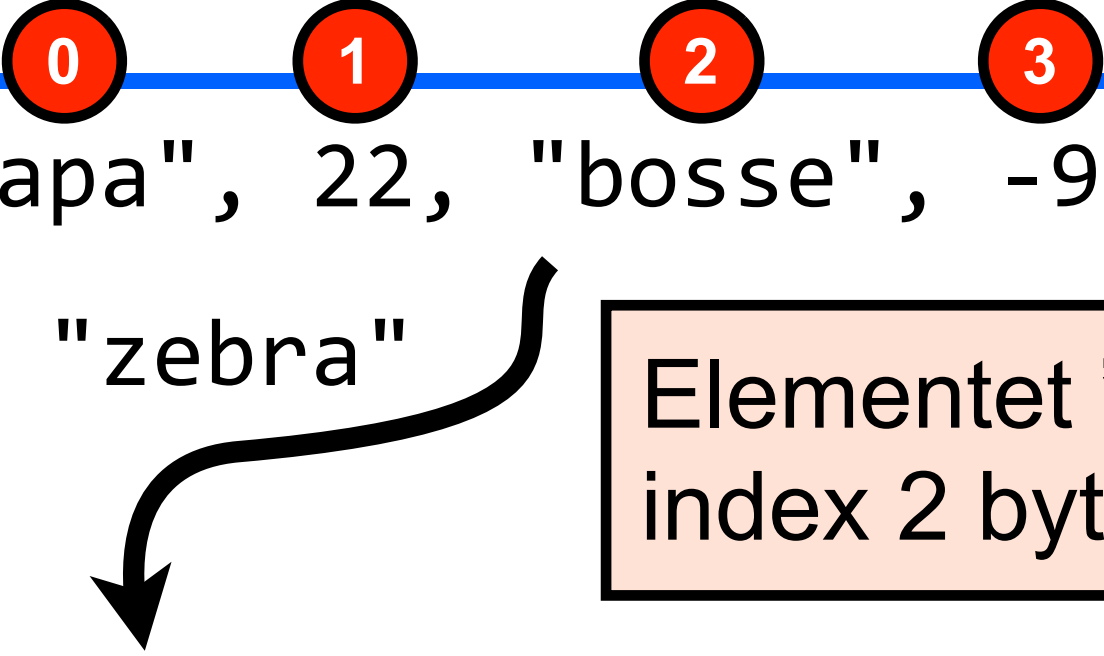
För att erhålla värdet på ett enskilt element i en lista används hakparenteser tillsammans med index .



```
>>> data = ["apa", 22, "bosse", -9]
>>> data[0]
'apa'
>>> data[2]
'bosse'
>>> data[-1]
-9
>>> data[-2]
'bosse'
```

Ändra värdet på ett element i en lista

För att ändra värdet på ett enskilt element i en lista används hakparenteser tillsammans med index och tilldelning med ett likhetstecken.



```
>>> data = ["apa", 22, "bosse", -9]
>>> data[2] = "zebra"
>>> data
['apa', 22, 'zebra', -9]
```

Elementet "bosse" på index 2 byts ut till "zebra".

Lägga till ett nytt värde sist i en lista

För att lägga till ett nytt element sist i en lista används metoden `append()`.

```
>>> alfa = ['a', 'b', 'c']
```

```
>>> alfa.append('d')
```



namn på
variabel med
en **lista**.

punkt

element att lägga till
sist i listan.

```
>>> alfa          # Elementet 'd' har lagts till sist i listan alfa.  
['a', 'b', 'c', 'd']
```

Skapa listor med range()

Den inbyggda funktionen `range()` kan användas för att skapa listor. Använd `help()` för att lära dig hur `range()` fungerar.

```
>>> help(range)
```

```
Help on built-in function range in module __builtin__:
```

```
range(...)
```

```
range(stop) -> list of integers
```

```
range(start, stop[, step]) -> list of integers
```

Return a list containing an arithmetic progression of integers.

`range(i, j)` returns `[i, i+1, i+2, ..., j-1]`; start (!) defaults to 0.

When step is given, it specifies the increment (or decrement).

For example, `range(4)` returns `[0, 1, 2, 3]`. The end point is omitted!

These are exactly the valid indices for a list of 4 elements.

```
(END)
```

Tryck **Q** eller **q** för att komma tillbaka till Python-prompten.

Skapa listor med range()

Den inbyggda funktionen `range()` kan användas för att skapa listor. Använd `help()` för att lära dig hur `range()` fungerar.

```
>>> help(range)
```

```
>>> range(10)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> range(3, 11)
```

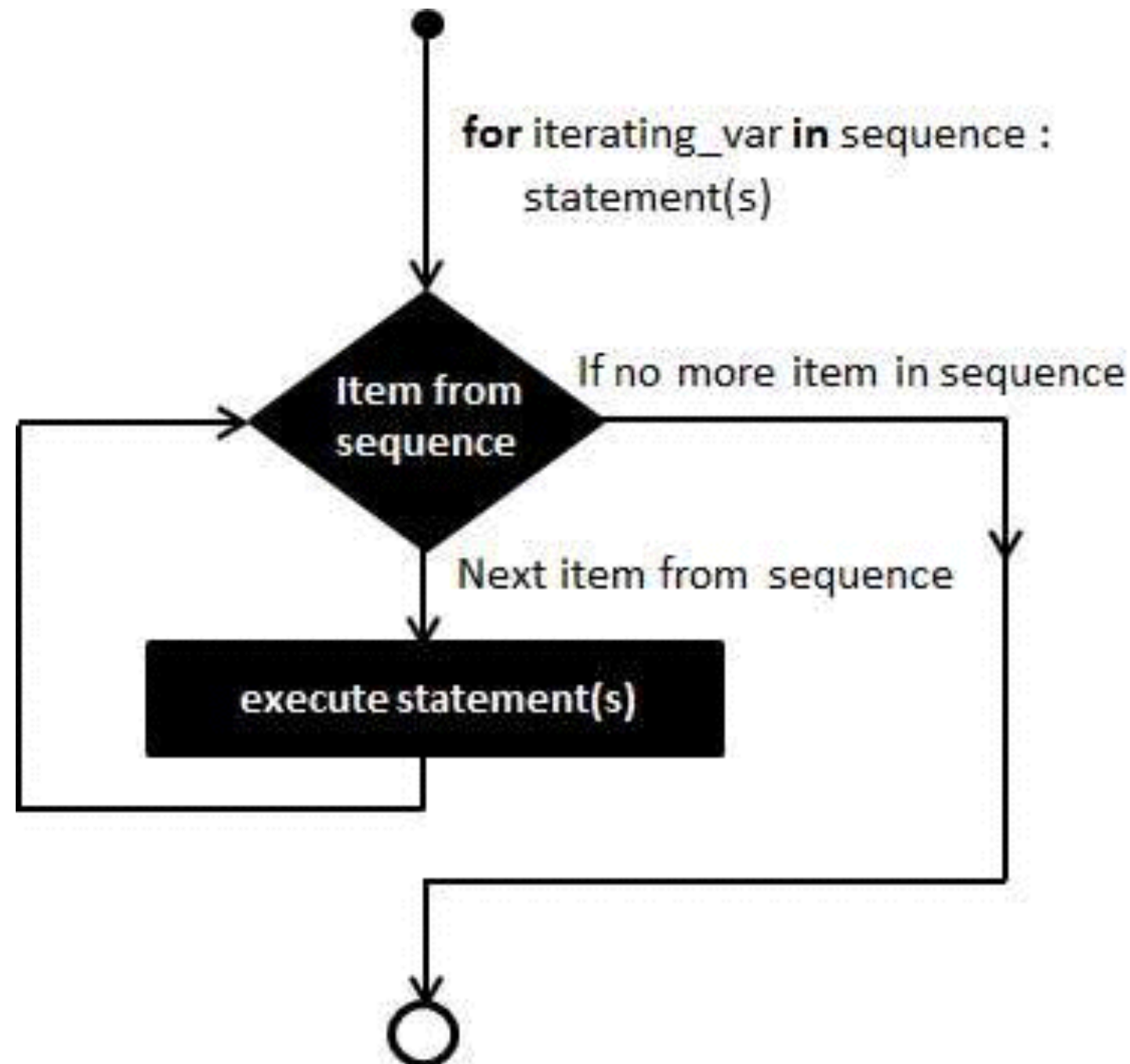
```
[3, 4, 5, 6, 7, 8, 9, 10]
```

```
>>> range(2, 10, 2)
```

```
[2, 4, 6, 8]
```

Slinga över alla element i en lista

För att konstruera slingor över listor i Python används kontrollstrukturen **for-in**.



Slinga över alla element i en lista

För att konstruera slingor över listor i Python används kontrollstrukturen **for-in**.

variabel lista kolon

```
>>> for x in [1,2,3]:  
...     print x  
...  
1  
2  
3  
>>>
```

Listan stegas igenom från vänster till höger. För varje varv i slingan tilldelas variabeln **x** ett nytt värde ur listan.

Det som skall utföras för varje element i listan måste skrivas efter ett indrag (tryck tab en gång).

Python Tutor

Experimentera med slingan **for-in** i Python tutor.

```
→ 1 kompisar = ["Eva", "Magnus", "Brasse"]  
2  
3 for kompis in kompisar:  
4     print kompis
```



Länk till **Python Tutor** med detta exempel finns på sidan **Del 2 - workshop** ➤ **Python Tutor**.

for-in är ett specialfall av while

```
→ 1  kompisar = ["Eva", "Magnus", "Brasse"]
   2
   3  print "== while =="
   4
   5  i = 0 # index.
   6
   7  while i < len(kompisar):
   8      print kompisar[i]
   9      i = i + 1
  10
  11  print "== for-in =="
  12  |
  13  for kompis in kompisar:
  14      print kompis
```

Länk till **Python Tutor** med detta exempel finns på sidan **Del 2 - workshop** ➤ **Python Tutor**.

Namn på variabler

(1)

Det finns många olika system för att namnge variabler. Låt oss skapa ett system för hur vi namnger våra variabler.

I de fall variabeln representerar något specifikt försöker vi ge variabeln ett beskrivande namn.

Exempel:

```
summa = 0
```

```
kompisar = ["Eva", "Magnus", "Brasse"]
```

Om det tydligt framgår av sammanhanget kan förkortningar användas.

```
s = 0          # s för summa.
```

Namn på variabler

(2)

Det finns många olika system för att namnge variabler. Låt oss skapa ett system för hur vi namnger våra variabler.

I de fall variabeln representerar något icke-specifikt försöker vi ge variabeln ett generellt och kort namn.

Exempel:

Variabelnamn	Betydelse
n	Ett godtyckligt tal (number), kan vara en int eller en float.
ns	En lista med godtyckliga tal (numbers).
x	Ett godtyckligt värde där vi inte bryr oss om det är ett heltal, flyttal, en sträng eller något annat.
xs	En lista med godtyckliga värden där vi inte bryr oss om vilken datatyp de olika elementen i listan har.

Beräkna längden av en lista

Om du **inte** får använda den inbyggda funktionen `len()` hur kan du då skriva en funktion som returnerar längden av en lista?

```
→ 1 def length(xs):  
2     '''Returns the number of elements in the list xs'''  
3  
4     n = 0  
5  
6     # TODO: Add code here to update n ...  
7  
8     return n  
9
```

Länk till **Python Tutor** med detta exempel finns på sidan **Del 2 - workshop** ➤ **Python Tutor**.

None

Ibland behöver man ett värde som betyder **ingenting** eller **okänt**, då används värdet **None** i Python.

None är endast lika med None. Notera att alla andra värden är större än None.

```
>>> None == None
True
>>> None != None
False
>>> None == 127
False
>>> None != 127
True
```

```
>>> None < None
False
>>> None > None
False
>>> 127 > None
True
>>> None > 127
False
```

Hitta det största talet i en lista

Om du **inte** får använda den inbyggda funktionen `max()` hur kan du då skriva en funktion som returnerar det största talet i en lista genom att använda kontrollstrukturen `for-in`?

```
→ 1 def max(ns):  
2     m = None  
3  
4     # TODO: Update m here ...  
5  
6     return m  
7  
8 ns = [3, 1, 4, 7, 3, -5]  
9 result = max(ns)
```

Länk till **Python Tutor** med detta exempel finns på sidan **Del 2 - workshop** ➤ **Python Tutor**.


Kopiera hela eller delar av en lista (1)

För att skapa en kopia av en lista eller delar av en lista används **slice-notation** `[start : (stop+1)]` vilket betyder att kopian skall innehålla alla element mellan index start och index stop.

```
>>> alfa = ['a', 'b', 'c']
>>> beta = alfa[:]      # En kopia av alla element i alfa.
>>> alfa.append('d')    # Lägg till 'd' sist i listan alfa.
>>> alfa
['a', 'b', 'c', 'd']   # Listan alfa är uppdaterad ...
>>> beta
['a', 'b', 'c']        # men inte beta som är en kopia.
```

Kopiera hela eller delar av en lista (2)

För att skapa en kopia av en lista eller delar av en lista används **slice-notation** `[start : (stop+1)]` vilket betyder att kopian skall innehålla alla element mellan index start och index stop.



A horizontal blue line represents a list. Above the line, five red circles with black outlines contain the indices 0, 1, 2, 3, and 4 from left to right.

```
>>> ns = [1, 2, 3, 4, 5]
>>> ns1 = ns[0:2]      # En kopia från index 0 till 1 (2-1).
>>> ns1
[1, 2]
>>> ns2 = ns[1:4]      # En kopia från index 1 till 3 (4-1).
>>> ns2
[2, 3, 4]
>>> ns3 = ns[2:]       # En kopia från index 2 till sista elementet.
>>> ns3
[3, 4, 5]
```


Strängar och tecken

Strängar används för att lagra och manipulera text. En sträng består av ett antal olika tecken och omges av enkla eller dubbla citattecken.

Strängar påminner en hel del om listor. I Python fungerar mycket ungefär likadant för strängar som det gör för listor.

Längden på en sträng

Precis som för listor används `len()` för att ta reda på längden (antal tecken) i en sträng.

```
>>> a = ""
```

```
>>> b = "A string I am!"
```

```
>>> len(a)
```

```
0
```

```
>>> len(b)
```

```
14
```

Index

Elementen i en sträng indexeras så att det första tecknet har index 0, det andra tecknet index 1 osv. Med hjälp av negativa index kan strängen indexeras baklänges.

```
>>> s = '0123456789ABCDEF'
```

```
>>> s[0]
```

```
'0'
```

```
>>> s[1]
```

```
'1'
```

```
>>> s[11]
```

```
'A'
```

```
>>> s[-1]
```

```
'F'
```

```
>>> s[-2]
```

```
'E'
```

```
>>> s[-7]
```

```
'9'
```

Kopiera hela eller delar av en sträng

För att skapa en kopia av en sträng eller delar av en sträng används **slice-notation** `[start : (stop+1)]` vilket betyder att kopian skall innehålla alla tecken mellan index start och index stop.

```
>>> s = '0123456789ABCDEF'
>>> s[:]           # Alla tecken från första till sista.
'0123456789ABCDEF'
>>> s[0:4]         # Från index 0 till 3 (4-1).
'0123'
>>> s[3:8]         # Från index 3 till 7 (8-1)
'34567'
>>> s[3:]          # Från index 3 till sista.
'3456789ABCDEF'
>>> s              # Notera att strängen s är oförändrad.
'0123456789ABCDEF'
```

Slinga över alla tecken i en sträng

För att konstruera slingor över strängar i Python används kontrollstrukturen **for x in string**.

```
>>> s = "Print me!"
>>> for c in s:
...     print c
...
P
r
i
n
t

m
e
!
```

Ändra värdet på ett tecken i en sträng

I motsats till listor går det inte att ändra värdet på ett element (tecken) i en sträng.

Elementen i en lista kan uppdateras.

```
>>> xs = ["apa", 22, "bosse"]  
>>> xs[1] = 777  
>>> xs  
['apa', 777, 'bosse']
```

Tecknen i en sträng inte uppdateras.

```
>>> s = "A string I am!"  
>>> s[0] = 'X'  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: 'str' object does not support item  
assignment
```

Räckvidd (scope)

Vad menas med begreppet räckvidd?

In computer programming, the **scope** of a name binding – an association of a name to an entity, such as a **variable** – is the **part of a computer program where the binding is valid**: where the name can be used to refer to the entity.

In other parts of the program the name may refer to a different entity (it may have a different binding), or to nothing at all (it may be unbound).

```

1 x = 55
2
3 def foo(y):
4     x = 2 * y
5     return x - 1
6
7 z = foo(4)
8
9 print z

```

Rad	Variabel	Räckvidd
1	x	?
3	y	?
4	x	?
7	z	?


```
1 x = 55
2
3 def foo(y):
4     x = 2 * y
5     return x - 1
6
7 z = foo(4)
8
9 print z
```

Rad	Variabel	Räckvidd
1	x	global
3	y	lokal
4	x	lokal
7	z	global

Detta **y** är en parameter och blir en **lokal variabel** vid funktionsanrop.

Detta **x** är en **lokal variabel** som endast existerar när funktionen foo() körs.

```

1  x = 55
2
3  def foo(y):
4      x = 2 * y
5      return x - 1
6
7  z = foo(4)
8
9  print z

```

Rad	Variabel	Räckvidd
1	x	global
3	y	lokal
4	x	lokal
7	z	global

Frames

Objects

Global frame

x 55
foo

function
foo(y)

foo

y 4
x 8

Lokala variabler
inom funktionen
foo().

Den lokala variabeln **y** skapas och får sitt värde vid funktionsanrop. I detta exempel anropas foo() med argumentet 4 och **y** skapas och tilldelas värdet 4.

Sidoeffekt

Vad menas med begreppet sidoeffekt?

Med sidoeffekter avses inom datorprogrammering och datalogi **effekter** vid exempelvis **funktionsanrop** som **inte är uppenbara** att de ska inträffa. Sidoeffekter kan vara önskade och medvetet införda, eller oönskade och omedvetet införda.

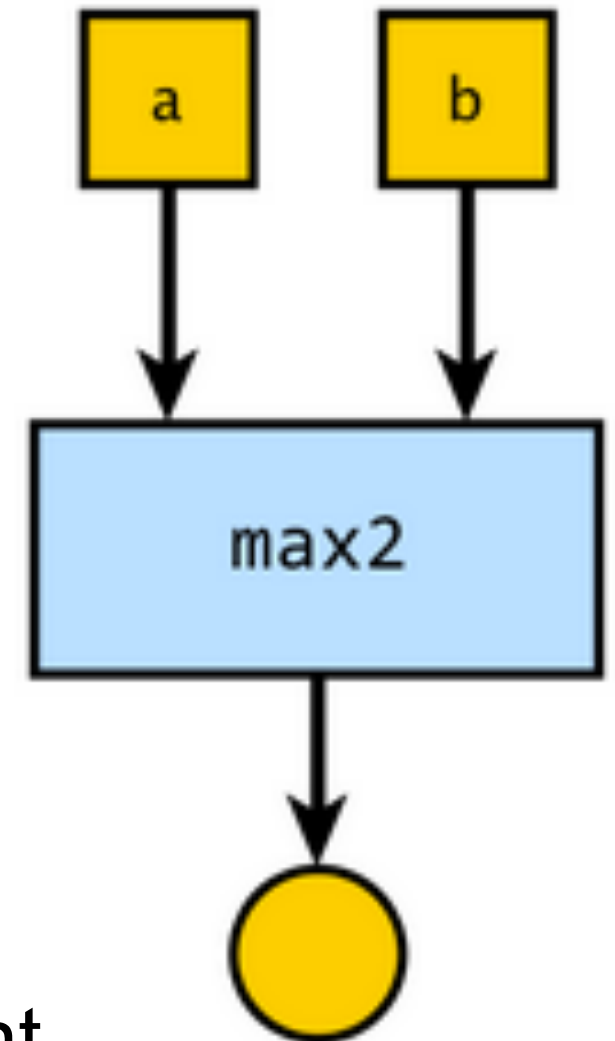
Ett vanligt exempel är att variabelvärden som andra delar av programvaran är beroende av, globala variabler, ändras.

Det anses vara dålig programmeringssed att införa sidoeffekter eftersom de drabbar programvarans överskådlighet och dessutom ofta inför felaktigheter i programvarans funktion.

Sidoeffekt

En funktion som påverkar något förutom att returnera ett eller flera värden sägs ha sidoeffekter.

```
def max2(a, b):  
    if a > b:  
        return a  
    else:  
        return b
```



Vid anrop av funktionen `max2` behöver anroparen inte känna till hur funktionen fungerar på insidan. Anroparen behöver endast tänka på vilka argument som används och vilket resultatet blir.

Funktionen `max2()` har inga sidoeffekter.

Sidoeffekt

En funktion som påverkar något förutom att returnera ett eller flera värden sägs ha sidoeffekter.

```
def hello(name):  
    print "Hello", name
```

Funktionen `hello()` tar ett argument, skriver ut till skärmen och returnerar ingen ting.

Funktionen `hello()` skriver till skärmen som en sidoeffekt.

Python tutor

Exempel i Python tutor som demonstrerar skillnaden mellan: **globala** och **lokala variabler**; **pure** functions och **non-pure** functions; funktioner med och utan **side-effekter**.

```
1  '''
2  Demonstrates the following concepts:
3
4  - Global and local variables.
5  - Pure functions.
6  - Local variables shadowing global variables.
7  - Non-pure functions.
8  '''
9
10 x = 55 # Global variable x.
11
12 def pure(n):
13     '''
14     Uses only local variables.
15
16     Note: A pure function only depends on its arguments
17         to calculate a return value.
18     '''
```

Länk till **Python Tutor** med detta exempel finns på sidan **Del 2 - workshop** ➤ **Python Tutor**.

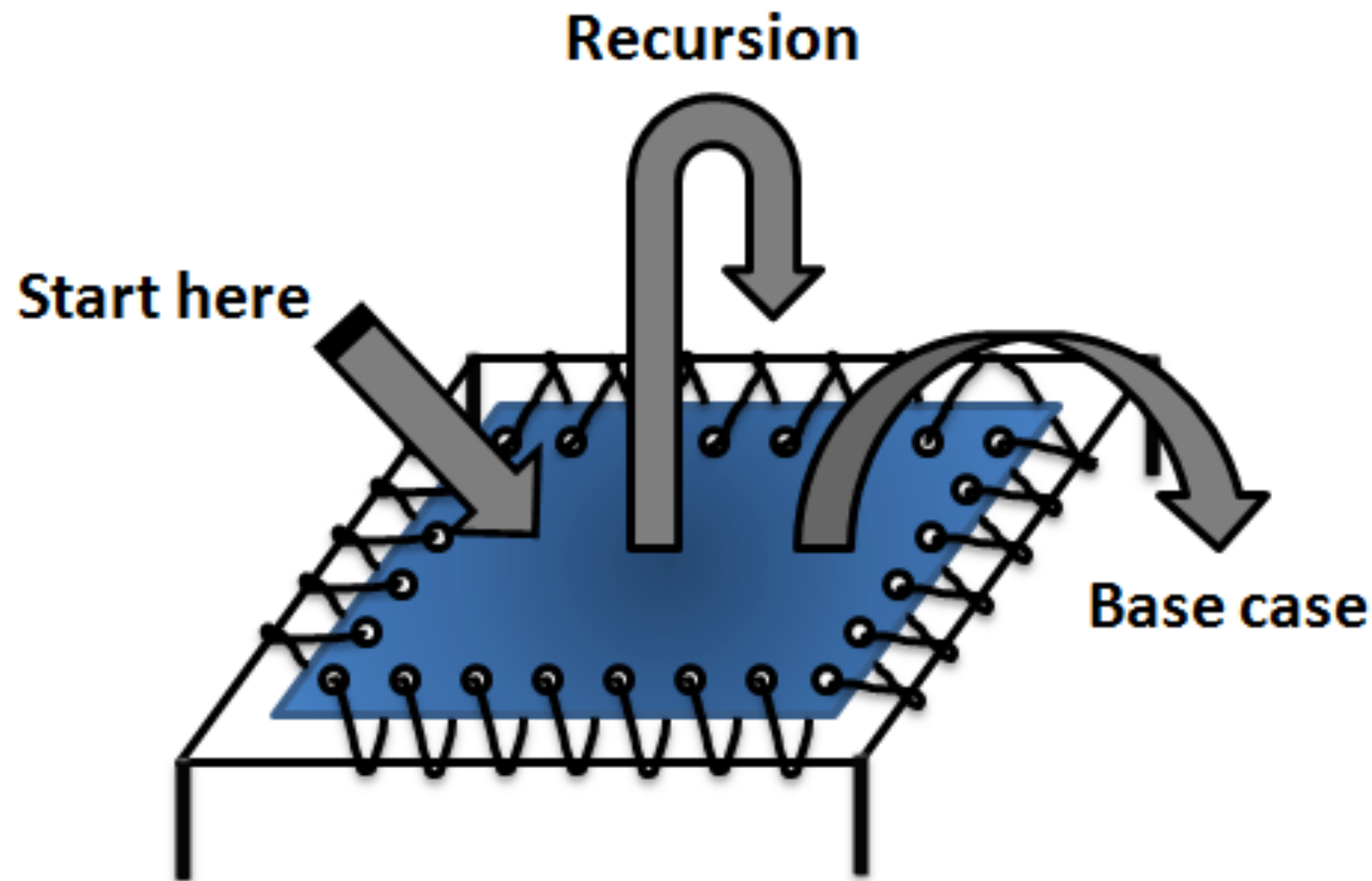
Obligatoriska uppgifter

I modulen [mandatory.py](#) hittar du ett antal funktioner som du skall färdigställa.



Frivilliga uppgifter

I modulen [optional.py](#) hittar du ett antal funktioner som du kan försöka färdigställa om du vill lära dig mer om **rekursion** och **högre ordningens funktioner**.



Högre ordningens funktioner

Funktioner som tar en eller flera andra funktioner som argument kallas för högre ordningens funktioner.

```
→ 1 def double(n):  
2     return 2*n  
3  
4 def triple(n):  
5     return 3*n  
6  
7 def mystery(f, g, x):  
8     return f(x) + g(x)  
9  
10 m1 = mystery(double, double, 3)  
11  
12 m2 = mystery(triple, double, 3)
```

Länk till **Python Tutor** med detta exempel finns på sidan **Del 2 - workshop** ➤ **Python Tutor**.

Söndra och härska

Ett kraftfullt verktyg vid problemlösning är att **separera det generella från det specifika**.

Ett sätt att åstadkomma detta är att låta en funktion ta en eller flera funktioner som parametrar. På detta sätt går det att fokusera på det generella beteendet och lämna det specifika beteendet till funktionerna som tas som argument och som på så sätt enkelt kan varieras.

```
>>> l = range(10)
>>> l
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> from mandatory import sum, is_even, is_odd
>>> from optional import filter
>>> sum(filter(is_even, range(10)))
20
>>> sum(filter(is_odd, range(10)))
25
```

Palindrom

En palindrom är en följd av tecken som blir likadan oavsett om man läser den framlänges eller baklänges.

Exempel på palindrom: **abba, girig, anna, otto**

I Guinness rekordbok anges den längsta palindromen, som består av ett enda ord, vara det finska ordet för tvål- eller lutförsäljare: **saippuakivikauppias** (19 bokstäver).

I dagligt tal avser man med palindromer hela fraser, till exempel: "**Mus rev inuits öra, sa röst i universum**", "**Ni talar bra latin**" eller "**Dromedaren Alpotto planerade mord**".

Ofta väljer man att bortse från blanksteg, skiljetecken och från skillnaden mellan små och stora bokstäver i fraser som är palindromer.

Rekursiv algoritm

Hur avgöra om en följd av tecken (sträng) är ett palindrom eller inte med hjälp av en rekursiv algoritm?



* Grundoperationer?

* Algoritm?

* Basfall?

Was it a car or a cat I saw?

Rekursiv algoritm

Hur avgöra om en följd av tecken (sträng) är ett palindrom eller inte med hjälp av en rekursiv algoritm?

```
→ 1 def is_palindrome(s):  
2     if len(s) < 2:  
3         return True  
4     else:  
5         first = s[0]  
6         last = s[-1]  
7         middle = s[1:-1]  
8  
9         if first == last:  
10            return is_palindrome(middle)  
11        else:  
12            return False
```

Denna lösning funkar bara för strängar utan skiljetecken och där alla bokstäver är gemener (eller versaler).

Länk till **Python Tutor** med detta exempel finns på sidan **Del 2 - workshop** ➤ **Python Tutor**.

Palindrom

Skriv en **rekursiv funktion** som avgör om en sträng är ett palindrom eller ej genom att använda dig av grundoperationerna (funktionerna) nedan.

Function	Description
<code>len(s)</code>	Returns the length of string s.
The following functions are defined in <code>utils.py</code>	
<code>clean(s)</code>	Returns a string equal to s but where all characters converted to lower case and all punctuation characters removed.
<code>first(s)</code>	Returns the first character in a string s.
<code>middle(s)</code>	Returns a string equal to s but with the first and last characters of s removed.
<code>last(s)</code>	Returns the last character in a string s.

Det är praktiskt att dela upp problemet i två funktioner. En funktion som "tvättar" strängen och sedan anropar den rekursiva funktionen som avgör om den tvättade strängen är ett palindrom eller inte.

```
def is_palindrome(s):  
    return is_palindrome_r(clean(s))  
  
def is_palindrome_r(s):
```

Rekursiv funktion

En rekursiv funktion anropar sig själv.

En rekursiv funktion behöver ett eller flera basfall.

Det är praktiskt att dela upp problemet i två funktioner. En funktion som *"tvättar"* strängen och sedan anropar den rekursiva funktionen som avgör om den tvättade strängen är ett palindrom eller inte.

```
def is_palindrome(s):  
    return is_palindrome_r(clean(s))  
  
def is_palindrome_r(s):  
    if len(s) < 2:  
        return True  
    elif first(s) == last(s):  
        return is_palindrome_r(middle(s))  
    else:  
        return False
```


Hitta det största talet i en lista

Om du **inte** får **använda** den inbyggda funktionen `max()` hur kan du då skriva en **rekursiv** funktion som returnerar det största talet i en lista genom att använda dig av grundoperationerna (funktionerna) nedan?

Dessa funktioner finns definierade i `utils.py`.

Function	Description
<code>empty(x)</code>	Returns True if the list or string x is empty and otherwise returns False.
<code>head(l)</code>	Returns the first element in list l. If l is empty, None is returned.
<code>tail(l)</code>	Returns a list equal to l but with the first element of l removed.

Det är praktiskt att dela upp problemet i två funktioner. En funktion som kontrollerar om det finns minst ett element i listan och i så fall anropar den rekursiva funktionen som hittar det största talet.

```
def max_r(l):  
    if len(l) > 0:  
        return max_r2(head(l), tail(l))  
  
def max_r2(acc, l):
```

Rekursiv funktion

En rekursiv funktion anropar sig själv.

En rekursiv funktion behöver ett eller flera basfall.

Det är praktiskt att dela upp problemet i två funktioner. En funktion som kontrollerar om det finns minst ett element i listan och i så fall anropar den rekursiva funktionen som hittar det största talet.

```
def max_r(l):  
    if len(l) > 0:  
        return max_r2(head(l), tail(l))  
  
def max_r2(acc, l):  
    if empty(l):  
        return acc  
    else:  
        h = head(l)  
  
        if h > acc:  
            return max_r2(h, tail(l))  
        else:  
            return max_r2(acc, tail(l))
```

Frivilliga uppgifter

I modulen **optional.py** hittar du ett antal funktioner som du kan försöka färdigställa om du vill lära dig mer om **rekursion** och **högre ordningens funktioner**.

Du kan nu fortsätta och färdigställa resterande funktioner i modulen **optional.py**.

Skriv dina lösningar direkt i filen **optional.py**.