

1 skärm = ett par ur basgruppen

Vilket par du tillhör framgår på kursens sida i **Studentportalen** >
Gruppindelningar > **Programmering 1 (par)**.

▼ GRUPPINDELNINGAR



Lektionsgrupper



Basgrupper



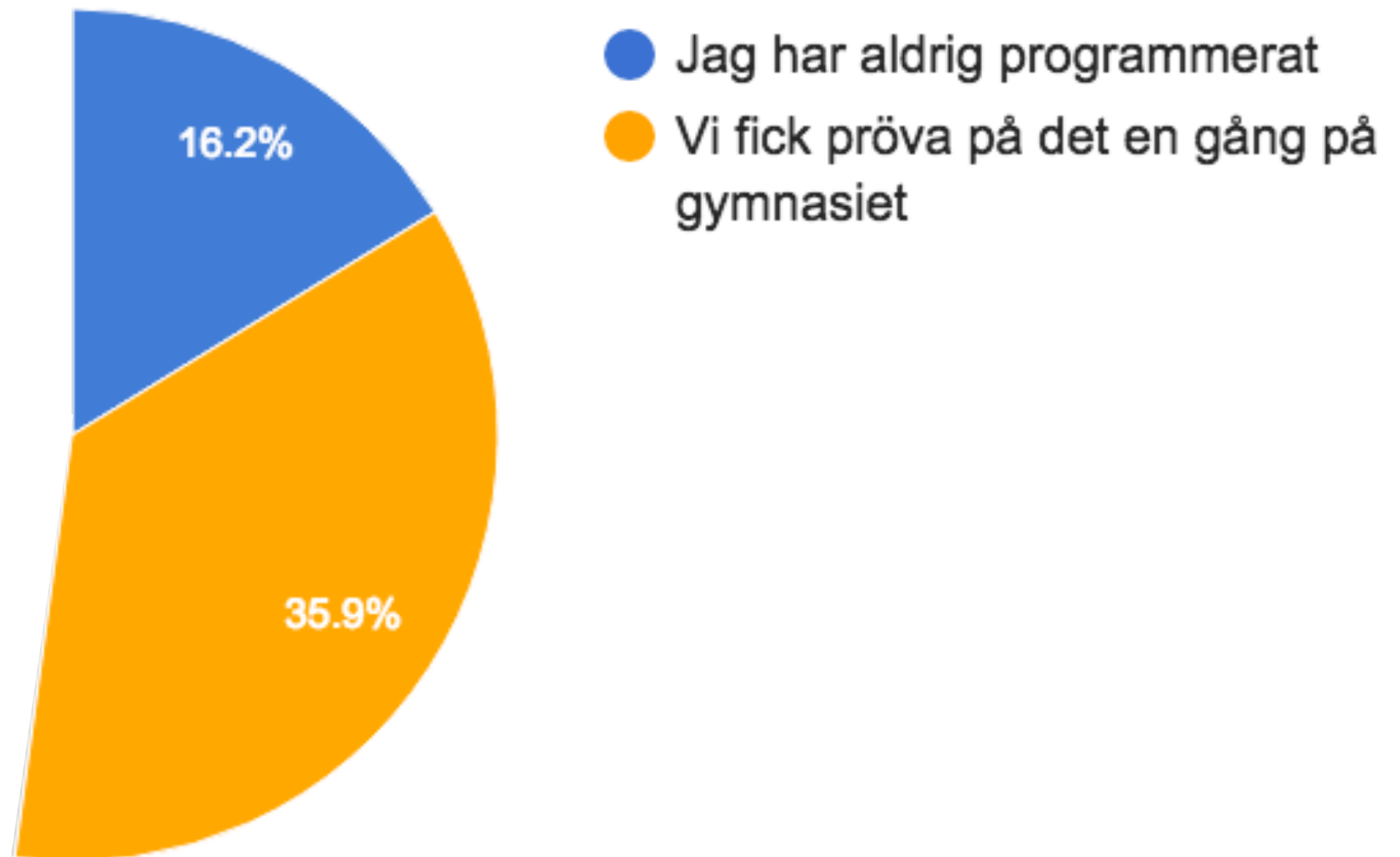
Programmering 1 (par)

Introduktion till informationsteknologi (1DT051)

Programmering (1) - wokshop

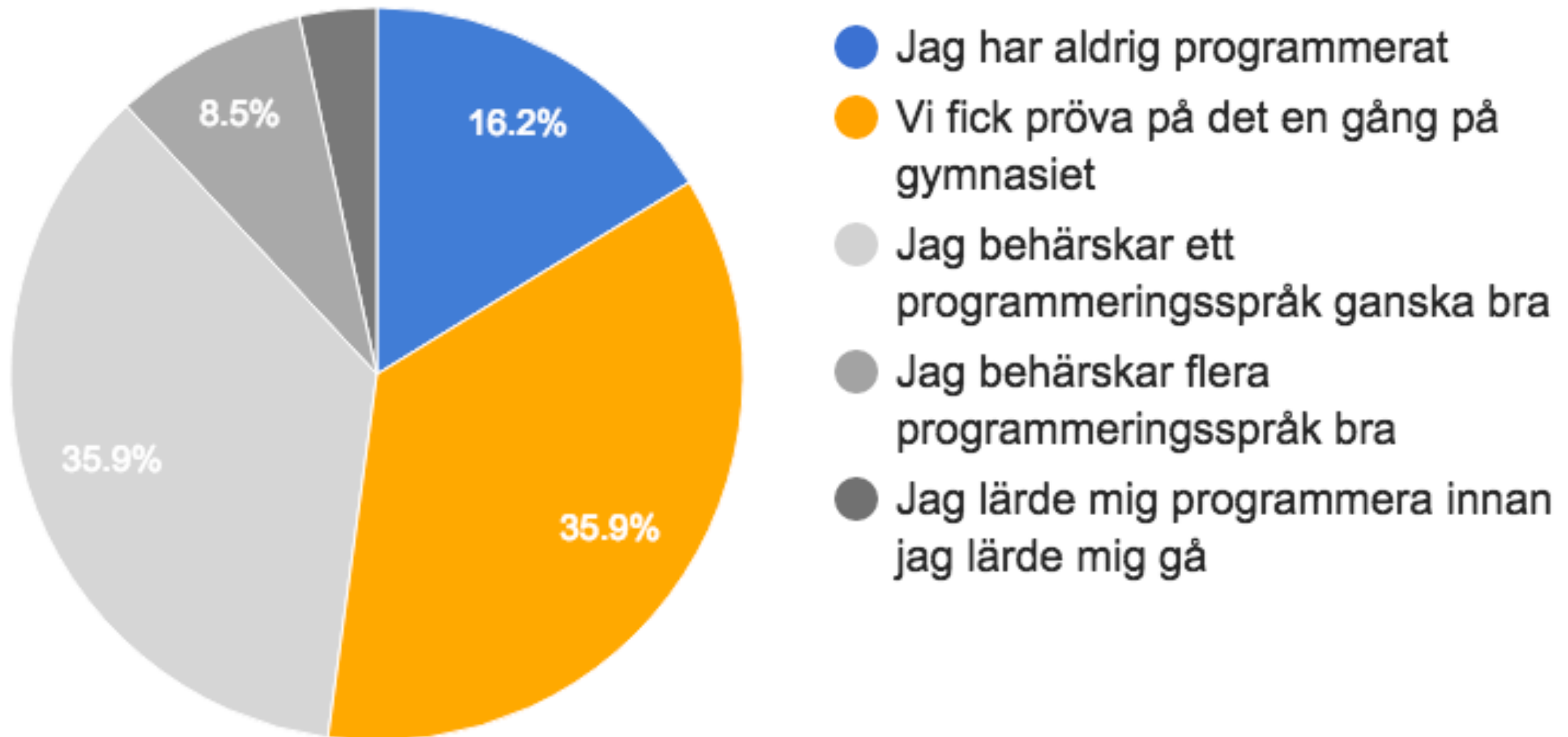
Tidigare erfarenheter av programmering

På första föreläsningen fick ni anonymt svara på ett antal frågor med hjälp av klickers (mentometer). Ett av resultaten visar att en mycket stor andel har liten eller ingen tidigare erfarenhet av programmering.



Tidigare erfarenheter av programmering

På första föreläsningen fick ni anonymt svara på ett antal frågor med hjälp av klickers (mentometer). Ett av resultaten visar att en mycket stor andel har liten eller ingen tidigare erfarenhet av programmering.



Algoritm

Vad menas med begreppet algoritm?

En **systematisk procedur** som i ett **ändligt antal steg** utför en beräkning eller löser ett givet problem.

En algoritm består av ett antal **operationer** och en **beskrivning** som anger i vilken ordning de olika stegen skall utföras.

Om vissa **förutsättningar** är uppfyllda leder metoden efter ett ändligt antal operationer till en lösning.

Programmering

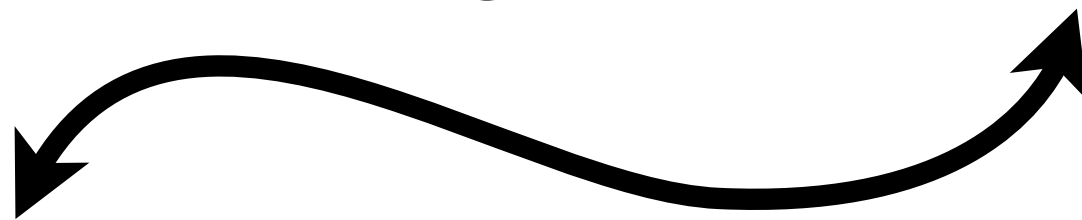
Vad menas med programmering?

Programmering handlar om att *instruera* en *maskin* eller del av en maskin, till exempel en mikrodator, dator, robot eller NC-maskin att *utföra* ett visst arbete.

Program

Vad menas med ett program?

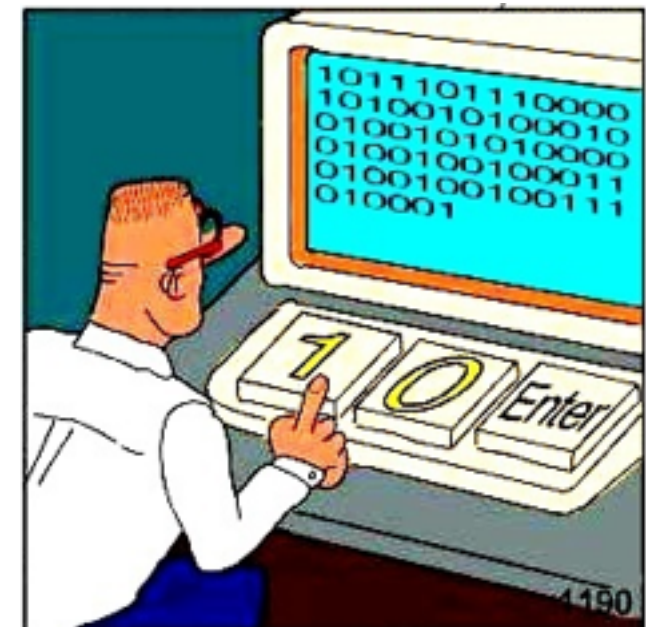
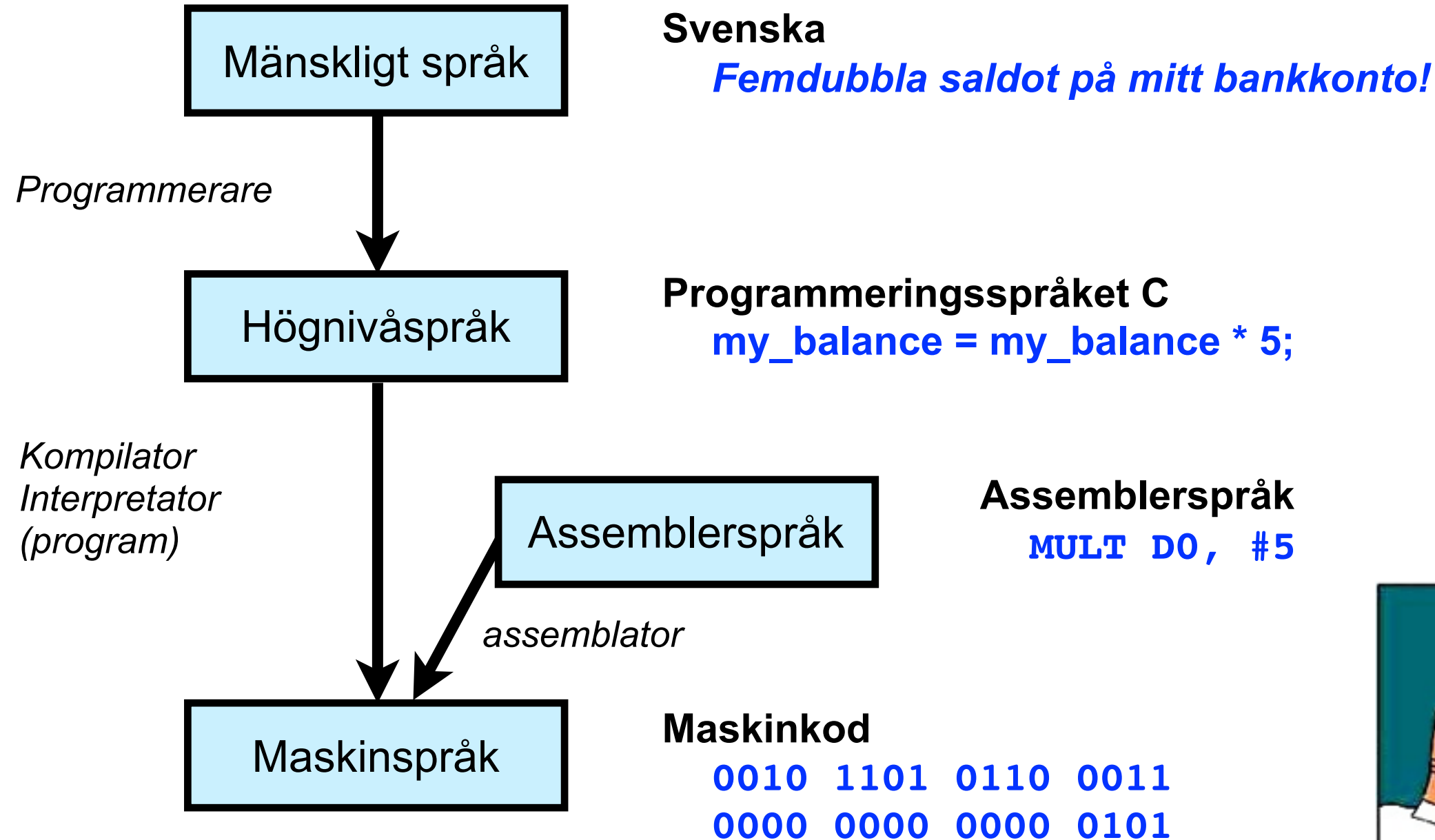
Ett datorprogram, även kallat dataprogram, är en serie *instruktioner* som *styr* en dator, och beskriver de *operationer* som datorn ska utföra, då programmet *körs*.



Att *exekvera* är det samma som att utföra, genomföra eller verkställa.

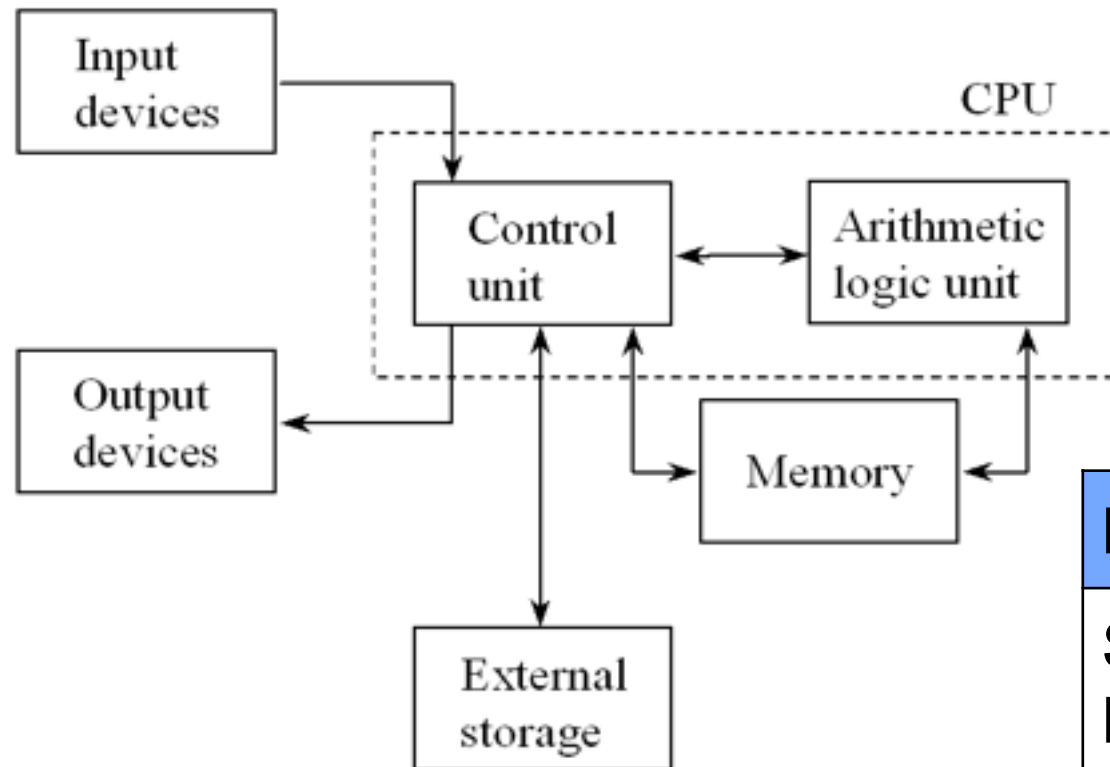
Datorns språk

Hur översätts ett problem till något som datorn förstår och kan utföra (exekvera).



Instruktionscykel

En dator hämtar och utför en instruktion i taget i en så kallad instruktionscykel.



Enhet(er)	Funktion
Styrenhet & Minnet	Hämta nästa instruktion från minnet.
Styrenhet	Avkoda instruktionen = översätt till interna styrsignaler.
Styrenhet & Minnet	Om instruktionen så kräver: hämta data att bearbeta från minnet.
ALU	Utför instruktionen
Styrenhet & Minnet	Om instruktionen så kräver: spara resultatet av instruktionen i minnet.

Interpretator vs kompilator

En *interpretator* eller *programtolk* är ett datorprogram som utför de aktiviteter som en viss programtext (så kallad *källkod*) beskriver. Detta till skillnad mot en *kompilator* som översätter programtexten till *maskinkod*.

För användaren av programmet tar det normalt längre tid att köra ett program med en interpretator än att köra motsvarande program som kompilerad maskinkod.

En interpretator kan dock spara tid för utvecklaren eller programmeraren eftersom den inte kräver det (ibland) tidskrävande kompileringssteget och därmed spar tid vid testning och avlusning av programsekvenser och algoritmer.

Python

Vi kommer att använda programmeringsspråket Python.

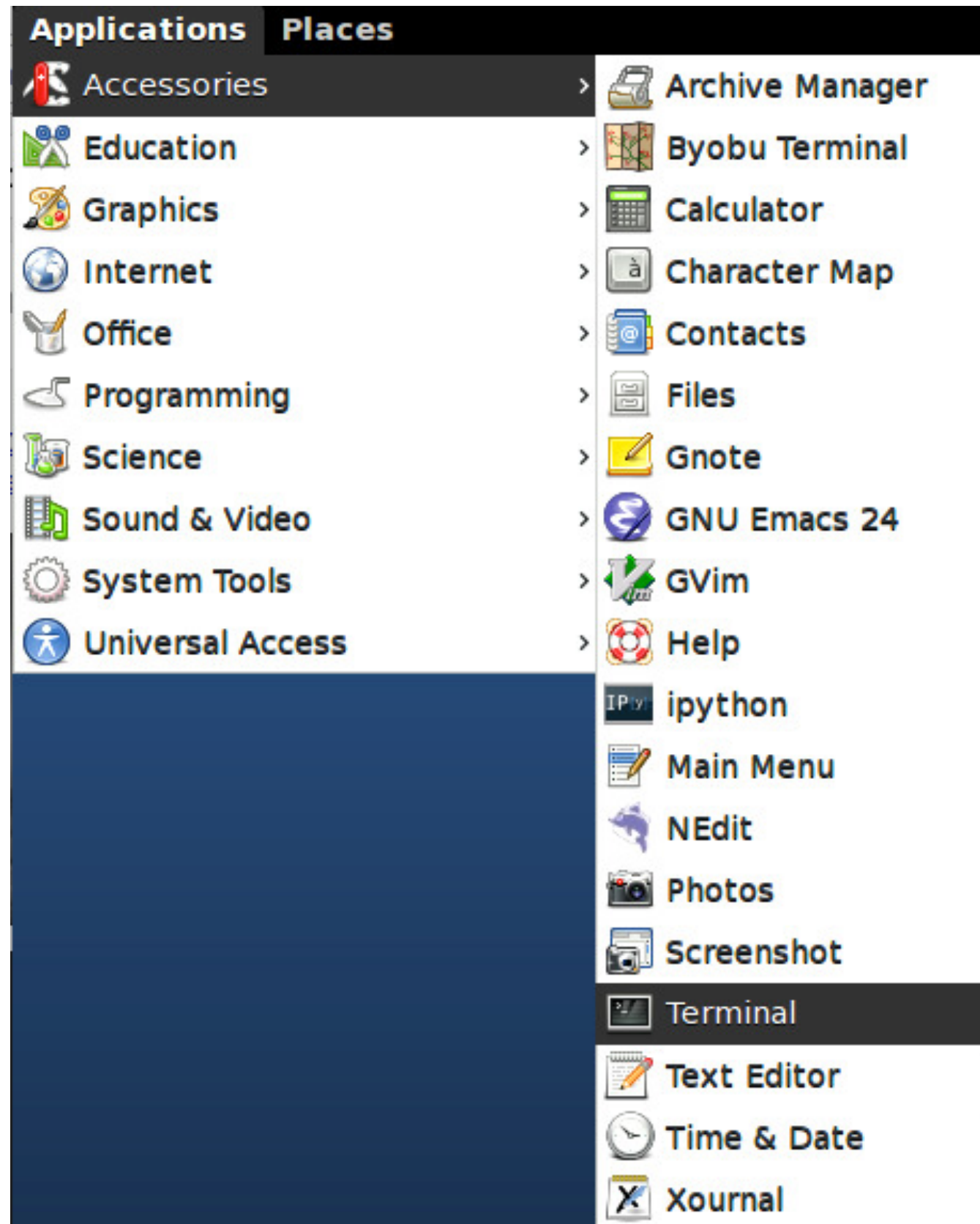
Python är (vanligen) ett *tolkat* (interpreterat språk).

Python är ett programspråk som utformades i slutet av 1980-talet av Guido van Rossum och som har fått sitt namn från Monty Python.



python

Starta en terminal



Starta en terminal

I terminalen bör du se en prompt liknande denna.

```
abcd1234@cronstedt:~$
```

Notera att istället för **abcd1234** bör du se ditt **användarnamn** och att du kan se ett annat **servernamn** än **cronstedt**.

I dessa instruktioner kommer följande prompt att användas för att ange att du skall skriva något i terminalen.

```
linux>
```

Starta Python

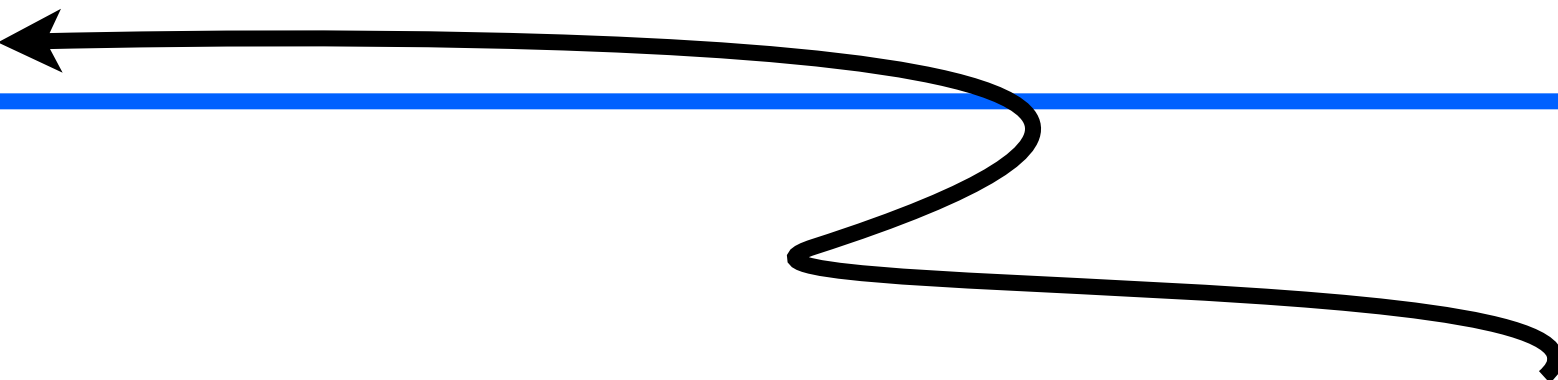
Skriv python vid kommando-prompten och tryck enter.

```
linux> python
```

Python-prompten

När du startat Python bör det se ut ungefär så här i terminalen.

```
linux> python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```



Notera att Python har en egen prompt >>>

Aritmetik

Aritmetik, *räknelära*, (från grekiskan arithmein: räkna, arithmetike: räknekonst, arithmos: tal) är den gren inom matematiken som behandlar räknande.

Det är den mest ursprungliga formen av matematik och innefattar grundläggande egenskaper hos *tal*, som hur de skrivs och hur de fungerar under *addition*, *subtraktion*, *multiplikation* och *division*; även andra räkne-operationer som *procenträkning*, *potenser*, *rotutdragnig* och *logaritmer* tillhör aritmetiken.

Aritmetiska operationer i Python

Testa att skriva några olika aritmetiska uttryck vid Python-prompten.

Operator



Addition	A + B
Subtraktion	A - B
Multiplikation	A * B
Division	A / B
???	A % B
???	A ** B

Prova att kombinera olika operatorer.

Testa att använda parenteser för att gruppera.

↑
↑
Operander

Observationer (aritmetik)

Adderar, subtraherar eller multiplicerar vi två heltal får vi ett nytt heltal.

Vid division av två heltal blir resultatet alltid ett heltal i Python.

Resten vid heltalsdivision fås med hjälp av operatoren för modulo (%).

XY** beräknar X upphöjt till Y.

Observationer (aritmetik)

Vid division av två heltal blir resultatet alltid ett heltal.

```
>>> 5 / 2
```

```
2
```

```
>>>
```

```
>>> 1 / 3
```

```
0
```

```
>>>
```

För att resultatet vid division skall bli ett decimaltal måste minst en av operanderna vara ett decimaltal.

```
>>> 5.0 / 2
```

```
2.5
```

```
>>>
```

```
>>> 5 / 2.0
```

```
2.5
```

```
>>>
```

```
>>> 5.0 / 2.0
```

```
2.5
```

```
>>>
```

Binära och hexadecimala tal

Vi kan skriva hexadecimala tal med hjälp av prefixet **0x**.

```
>>> 0xa
```

```
10
```

```
>>> 0x2b
```

```
43
```

Vi kan skriva binära tal med hjälp av prefixet **0b**.

```
>>> 0b101
```

```
5
```

```
>>> 0b1111
```

```
15
```

Binära och hexadecimala tal

Vi kan konvertera till binär form.

```
>>> bin(5)
'0b101'
>>> bin(0xf)
'0b1111'
```

Vi kan konvertera till hexadecimal form.

```
>>> hex(10)
'0xa'
>>> hex(43)
'0x2b'
```

Operatorn << (binärt shift åt vänster)

```
>>> 2 << 3
```

```
16
```

```
>>> bin(2 << 3)
```

```
'0b10000'
```

```
>>> 0b00010 << 3
```

```
16
```

```
>>> bin(0b00010 << 3)
```

```
'0b10000'
```

Logiska operationer i Python

Testa att skriva några olika logiska uttryck vid Python-prompten.

Lika med	A == B
Skillt 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

Prova att kombinera olika operatorer.

Testa att använda parenteser för att gruppera.

Observationer (logik)

Resultatet av en logiskt operation är antingen sant eller falskt. I Python representeras dessa två möjligheter av **True** och **False**.

En kombination av logiska operationer, ett logiskt uttryck, resulterar alltid i **True** eller **False**.

Kommentarer

I Python utgör allt efter tecknet # på en rad en kommentar och ignoreras av interpretatorn.

Du kan till exempel lägga till en kommentar direkt efter ett aritmetiskt uttryck.

```
>>> 1 + 1 # Skall bli två  
2  
>>>
```


Variabel

Vad menas med variabel?

En variabel är något som *kan ändras*.

Inom matematiken och datavetenskapen betecknar den ett *namngivet* objekt som används för att representera ett okänt värde, till exempel ett reellt tal.

Variabler används i öppna utsagor. De kan anses stå i motsats till konstanter som är oföränderliga, liksom till parametrar som hålls konstanta under en given process eller beräkning.

Variabel (programmering)

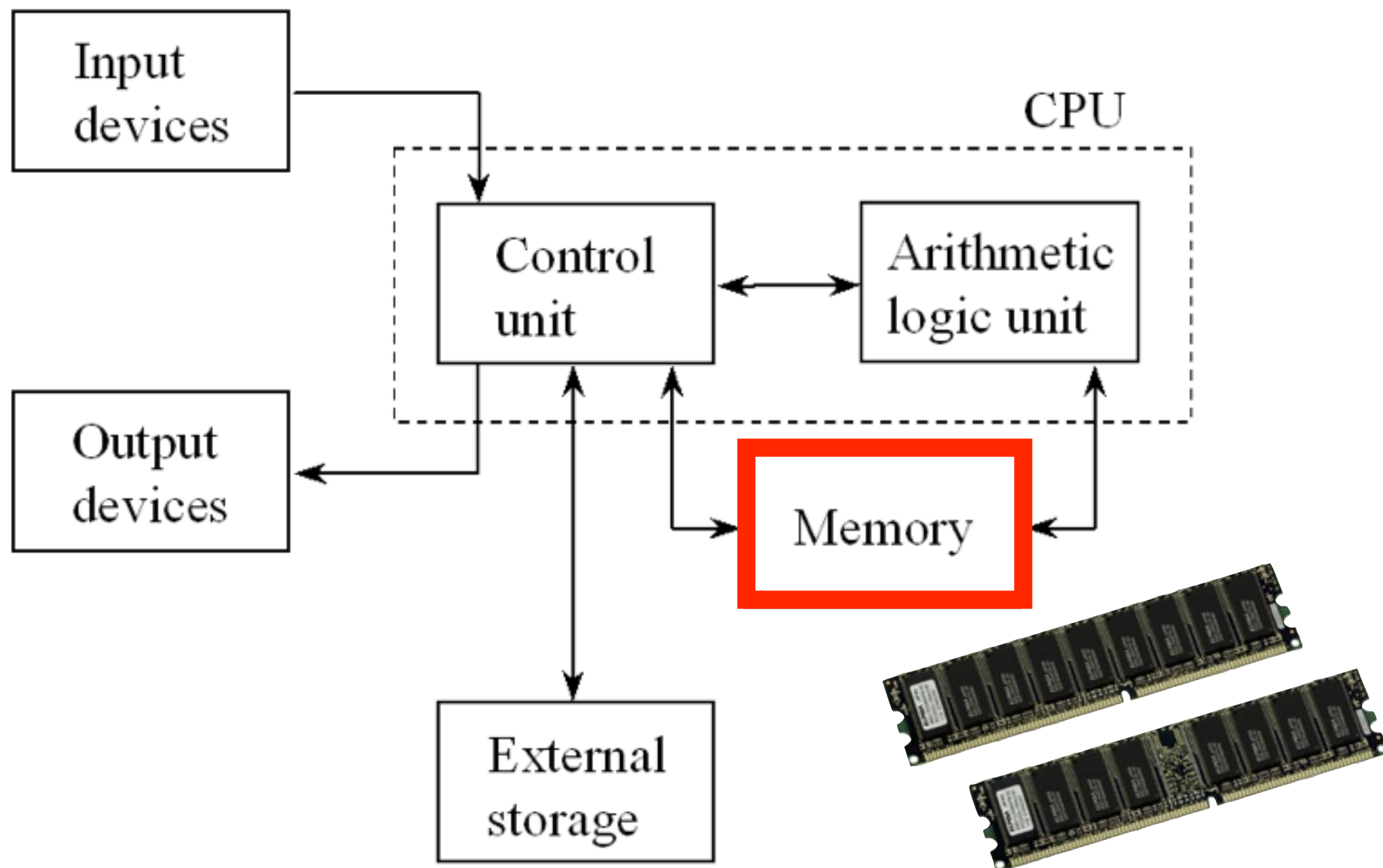
Vad menas med variabel i samband med programmering?

En variabel är något som *kan ändras*.

En variabel är ett sätt att *komma ihåg ett värde* genom att på en *namngiven* plats *lagra* detta värde.

Variabel (programmering och datorn)

En variabel är en namngiven plats i datorns minne där vi kan lagra värden.



Variabler (Python)

För att lagra ett värde i en variabeln används operatorn **=**.

Lagra värdet **127** i variabeln **a**.

```
>>> a = 127
```

Fråga Python vilket värde som lagrats i variabel **a**.

```
>>> a  
127  
>>>
```

Variabler (Python)

```
>>> a = 127      # Lagra värdet 127 i variabeln a.
>>> 2*a          # Utför beräkningar där ...
254
>>> 3*a          # variabeln a ingår.
381
>>> a            # Värdet på a har inte ändrats.
127
>>> a = a + 1    # Vad betyder detta?
>>> a            # Värdet på a har uppdaterats.
128
>>>
```

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 redan stött på två datatyper i Python.

```
>>> 5 / 2
```

```
2
```

```
>>>
```

```
>>> 5 / 2.0
```

```
2.5
```

```
>>>
```

- **int** - heltal ("integers" på engelska)

- **float** - flyttal (eller "reella tal")

Strängar i Python

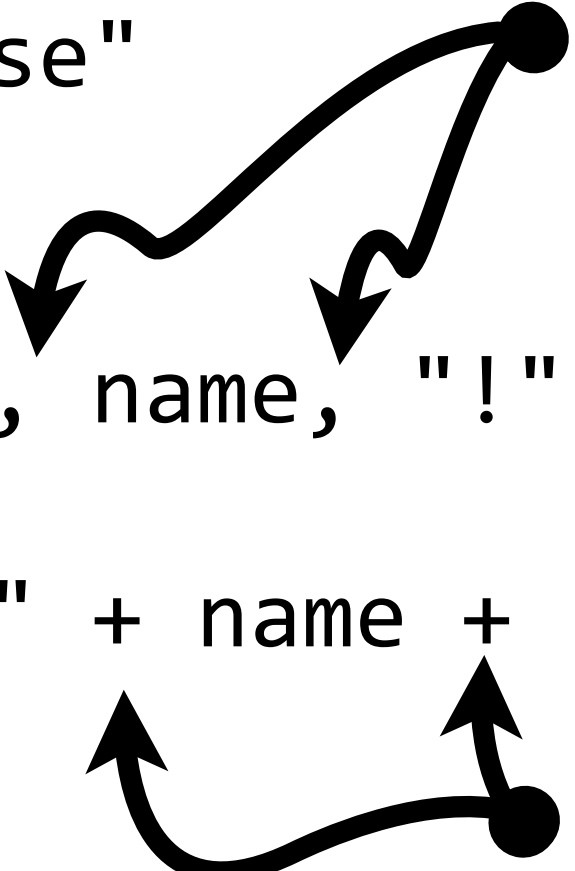
En sträng är en datatyp uppbyggd av tecken, såsom bokstäver, siffror och interpunktioner. I Python definieras den med apostrofer 'sträng' eller citationstecken "sträng".

```
>>> "Apa"  
'apa'  
>>> 'Bosse'  
'bosse'  
>>> "Apan Bosse"  
'Apan Bosse'
```


Skriva ut till skärmen

I Python kan vi skriva ut till skärmen med hjälp av print.

```
>>> print "Hej!"
Hej!
>>> name = "Bosse"
>>> print name
Bosse
>>> print "Hej", name, "!"
Hej Bosse !
>>> print "Hej " + name + "!"
Hej Bosse!
>>>
```

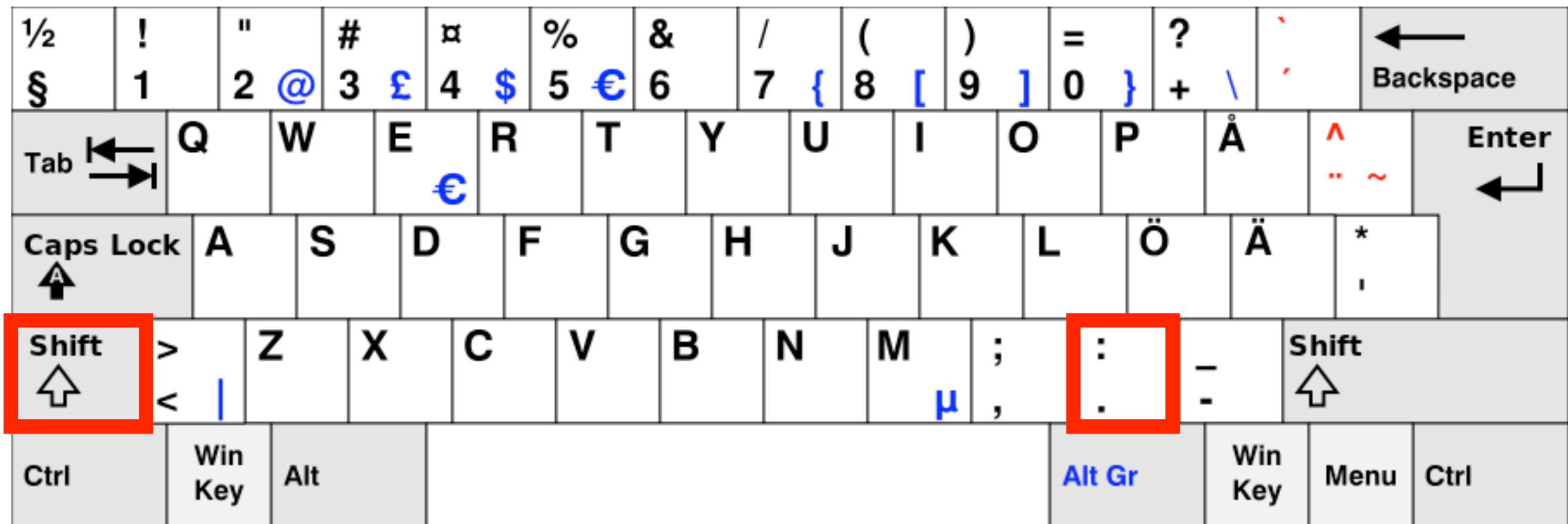


Kan rada up det vi vill skriva ut med hjälp av kommatecken. Varje sträng skrivs ut separerade av ett mellanslag.

Kan rada up det vi vill skriva ut med hjälp av plustecken. Vill vi ha mellanslag efter "Hej" måste detta läggas till som "Hej ".

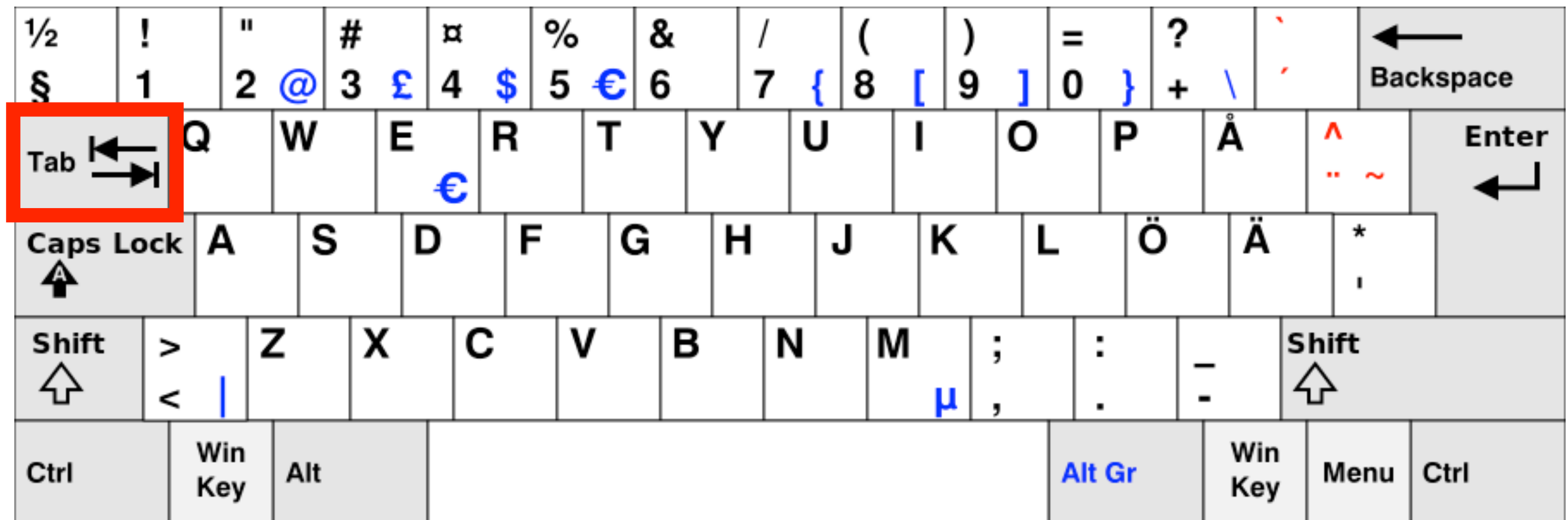
Kolon (:)

I Python används kolon (:) för att beskriva att det som följer efter kolon på något sätt hänger samman med det som kommer före kolon.



Tab (indrag)

I Python har indrag med tab betydelse för att beskriva hur block av kod hör ihop på olika sätt.



Testa villkor

I Python kan vi testa ett villkor och göra val med **if**.

Villkoret måste avslutats med kolon.

```
>>> x = 10
>>> if x > 100:
...     print "x is huge!"
...
>>>
```

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

I detta fall är villkoret falskt och inget skrivs ut.

Testa villkor

I Python kan vi testa ett villkor och göra val med **if**.

```
>>> x = 999
>>> if x > 100:
...     print "x is huge!"
...
x is huge!
>>>
```

Notera kolon (:) och indrag (tab).

I detta fall är villkoret sant och strängen "x is huge!" skrivs ut.

Testa villkor

Kan lägga till **else** för att utföra något om villkoret är falskt.

```
>>> x = 10
>>> if x > 100:
...     print "x is huge!"
...     else:
...     print "x is small!"
...
x is small!
>>>
```

Notera kolon (:) och indrag (tab).

I detta fall är villkoret falskt och strängen "x is small!" skrivs ut.

Testa flera villkor

Kan lägga till en eller flera **elif** för testa ytterligare ett eller flera villkor.

```
>>> x = 10
>>> 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!"
...
x is small!
>>>
```

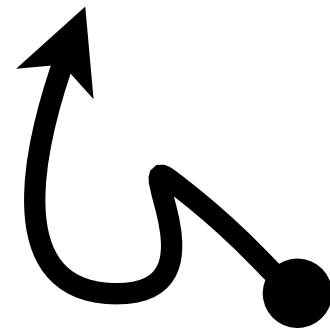
Notera kolon (:) och indrag (tab).

Uttryck

Vad menas med begreppet uttryck?

Ett uttryck är i matematik en **kombination** av **siffror**, **operatorer**, **grupperande symboler** (som klammer och parentes) och/eller **variabler** ...

... **ordnade** på ett meningsfullt sätt så att uttrycket kan bli **utvärderat**.



Kallas även för **evaluerat**.

Aritmetiska uttryck i Python

Exempel på aritmetiska i Python

```
>>> 3 + 2 * 10 / 2
```

```
13
```

```
>>> 2 * (7 / 2) + 7 % 2
```

```
7
```

```
>>> (1*2**0) + (0*2**1) + (1*2**2)
```

```
5
```

```
>>> 0b101 + 0x2b
```

```
48
```

Logiska uttryck i Python

Exempel på logiska uttryck i Python

```
>>> 127 == 999
```

```
False
```

```
>>> 127 < 999
```

```
True
```

```
>>> 127 != 99
```

```
True
```

```
>>> not 127 < 999
```

```
False
```

Utvärdering av uttryck

Exempel på vad som kan ingå i olika uttryck.

- Aritmetiska uttryck
- Logiska uttryck
- Variabler
- Funktionsanrop (mer om funktioner längre fram)
- Ett uttryck kan i sin tur bestå av flera andra uttryck.

Uttryck utvärderas “**bottom-up**” (inre uttryck först).

Utvärdering av uttryck steg för steg

Uttryck utvärderas “**bottom-up**” (inre uttryck först).

Först utvärderas

$$10 / 2 \rightarrow 5$$

$$3 + (2 * \boxed{(10 / 2)})$$

Diagram illustrating the first step of evaluation. The innermost expression $(10 / 2)$ is highlighted with a red box and labeled with a circled '1'.

Sedan utvärderas

$$2 * 5 \rightarrow 10$$

$$3 + (\boxed{2 * 5})$$

Diagram illustrating the second step of evaluation. The expression $2 * 5$ is highlighted with a red box and labeled with a circled '2'.

Slutligen utvärderas

$$3 + 10 \rightarrow 13$$

$$\boxed{3 + 10}$$

Diagram illustrating the third step of evaluation. The entire expression $3 + 10$ is highlighted with a red box and labeled with a circled '3'.

Resultatet av hela
uttrycket

$$\boxed{13}$$

Utvärdering av *if-else* steg för steg

```
x = 5
```

```
if x % 2 == 1 :
```

```
    x = 2 * x
```

```
else:
```

```
    x = 2 * x - 1
```

Utvärdering av *if-else* steg för steg

x = 5

if **x** % 2 == 1 :

x = 2 * x

else:

x = 2 * x - 1

Utvärdering av *if-else* steg för steg

x = 5

if 5 **% 2 == 1 :**

x = 2 * x

else:

x = 2 * x - 1

Utvärdering av *if-else* steg för steg

x = 5

if **5 % 2 == 1 :**

x = 2 * x

else:

x = 2 * x - 1

Utvärdering av *if-else* steg för steg

x = 5

if 1 **== 1 :**

x = 2 * x

else:

x = 2 * x - 1

Utvärdering av *if-else* steg för steg

x = 5

if **1 == 1** :

x = 2 * x

else:

x = 2 * x - 1

Utvärdering av *if-else* steg för steg

x = 5

if

True

:

x = 2 * x

else:

x = 2 * x - 1

Utvärdering av *if-else* steg för steg

```
x = 5
```

```
if True :
```

```
    x = 2 * x
```

```
else:
```

```
    x = 2 * x - 1
```

Utvärdering av *if-else* steg för steg

```
x = 5
```

```
if True :
```

```
    x = 2 * 5
```

```
else:
```

```
    x = 2 * x - 1
```

Utvärdering av *if-else* steg för steg

```
x = 5
```

```
if True :
```

```
    x = 2 * 5
```

```
else:
```

```
    x = 2 * x - 1
```

Utvärdering av *if-else* steg för steg

```
x = 5
```

```
if True :
```

```
    x = 10
```

```
else:
```

```
    x = 2 * x - 1
```

www.python tutor.com

På denna sida finns ett mycket bra verktyg för att förstå hur Python fungerar.

```
>>> x = 10
>>> if x > 100:
...     print "x is huge!"
... else:
...     print "x is small!"
...
x is small!
>>>
```

En snabbänk med koden ovan i Python Tutor. Länken finns även i Studentportalen.

<http://tinyurl.com/qaawg6d>

Kapslar i Logisim

En kapsel i Logisim tar en eller flera inputs och kan generera ett eller flera outputs.

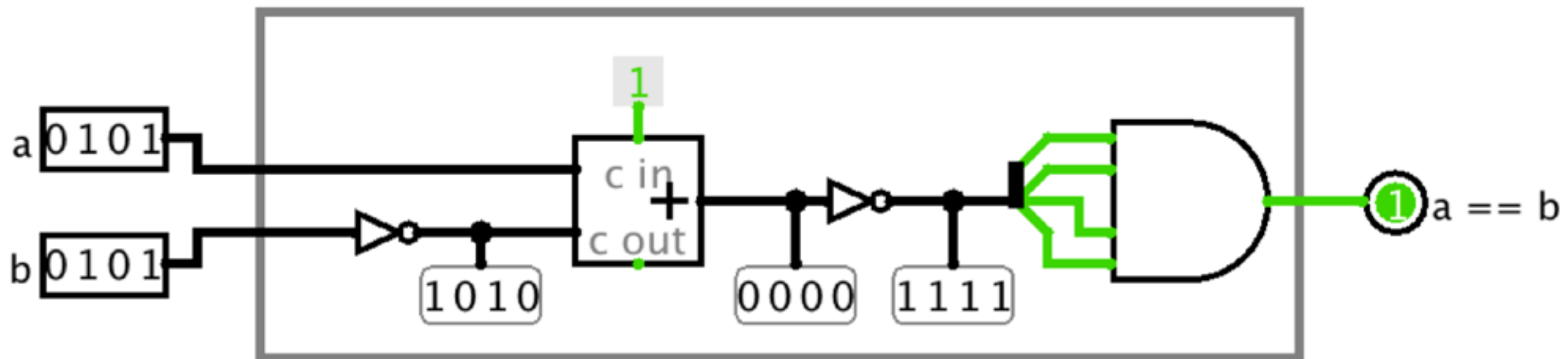
På bilden nedan ser vi hur kretsen equal tar två inputs **a** och **b** och beräknar en output **$a == b$** .



Kapslar i Logisim

En kapsel i Logisim tar en eller flera inputs och kan generera ett eller flera outputs.

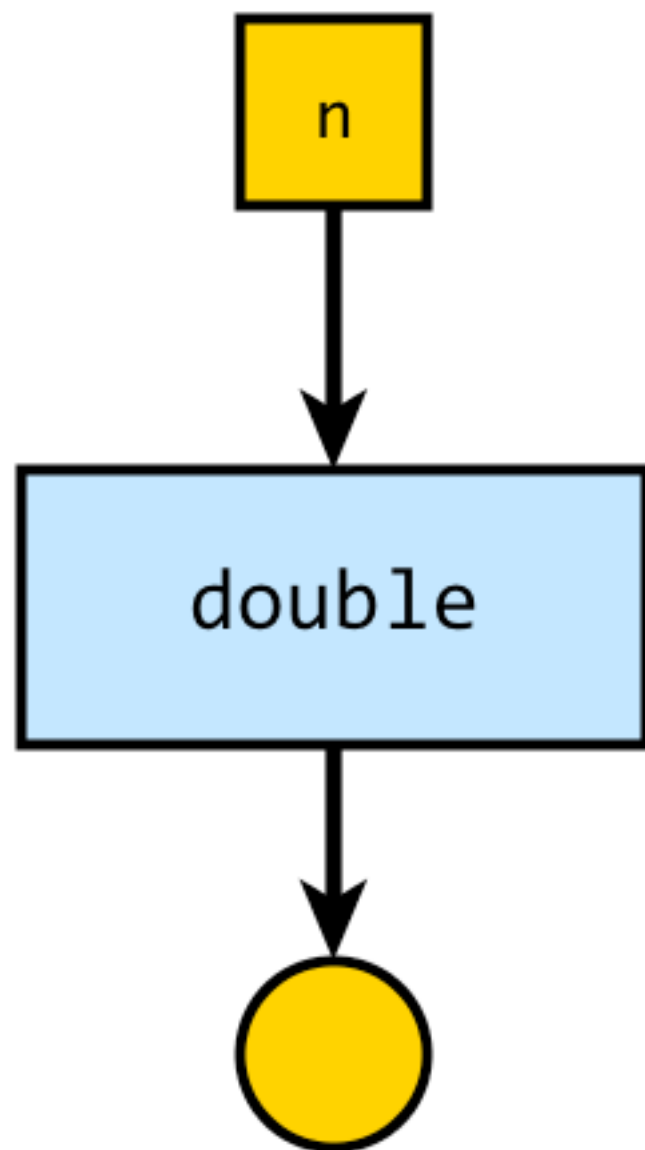
På bilden nedan ser vi hur kretsen equal tar två inputs **a** och **b** och beräknar en output **a == b**.



Funktioner i Python

Funktioner i Python påminner mycket om kapslar i Logisim

Parameter



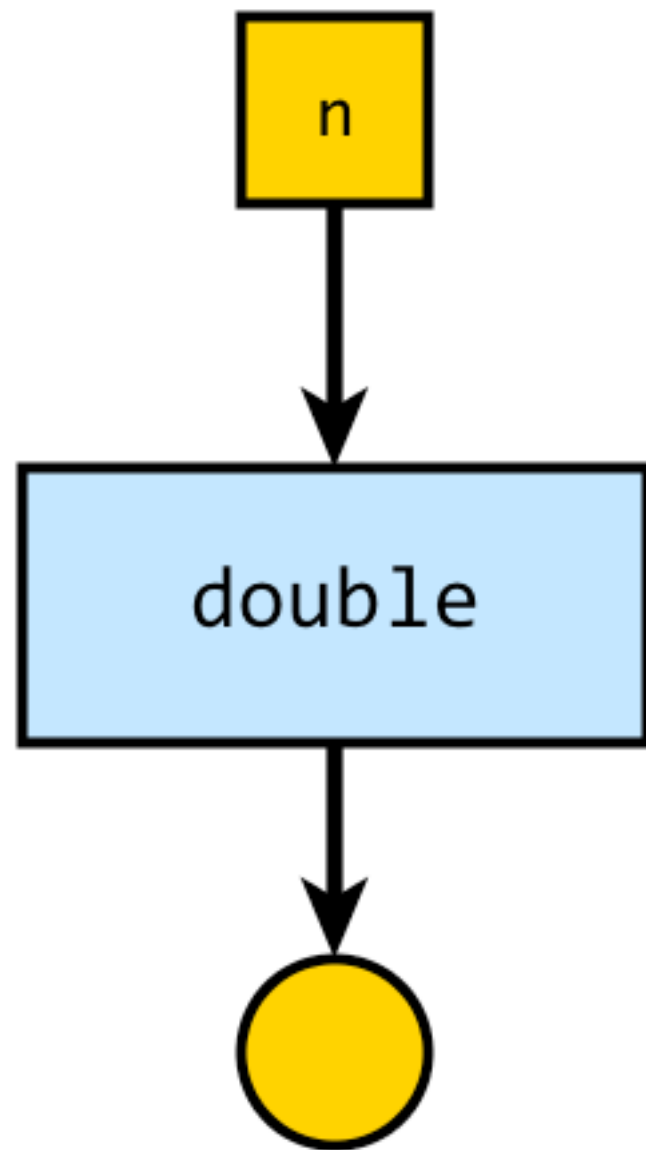
Returvärde (resultat)

En grafiskt representation av funktionen `double()`. Funktionen tar en parametrar `n` och och skall returnera det dubbla värdet `2*n`.

När vi anropar funktionen skall vi inte behöva känna till hur den bär sig åt för att beräkna resultatet.

Definiera funktioner i Python

Parameter



Returvärde (resultat)

Definition av funktionen **double()** i Python.

```
>>> def double(n):  
...     return 2*n  
...  
>>>
```

Notera kolon (:) och indrag (tab).

1 2 3 4

```
>>> def double(n):  
...     5 return 2*n  
...     6  
>>>
```

- 1) När vi definierar en funktion i Python måste vi börja med **def**.
- 2) Namnet på funktionen, i detta fall **double**.
- 3) Inom **parenteser** namnger vi de **parametrar** som funktionen beror av separerade av **kommatecken**, i detta fall den ensamma parametern n.
- 4) **Kolon**.
- 5) Resterande rader måste vara **indragna** (tryck tab).
- 6) Använder **return** för att berätta vilket resultatet av funktionen skall vara, i detta fall resultatet av uttrycket 2*n.

Anropa funktioner i Python

För att anropa en funktion skriver vi namnet på funktionen följt argument till funktionen inom parenteser.

```
>>> def double(n):  
...     return 2*n
```

```
...
```

```
>>>
```

7) >>> double(7)

8) 14

9) >>> double(3.5)

10) 7.0

```
>>>
```

7) Funktionen double **anropas** med **argumentet** 7.

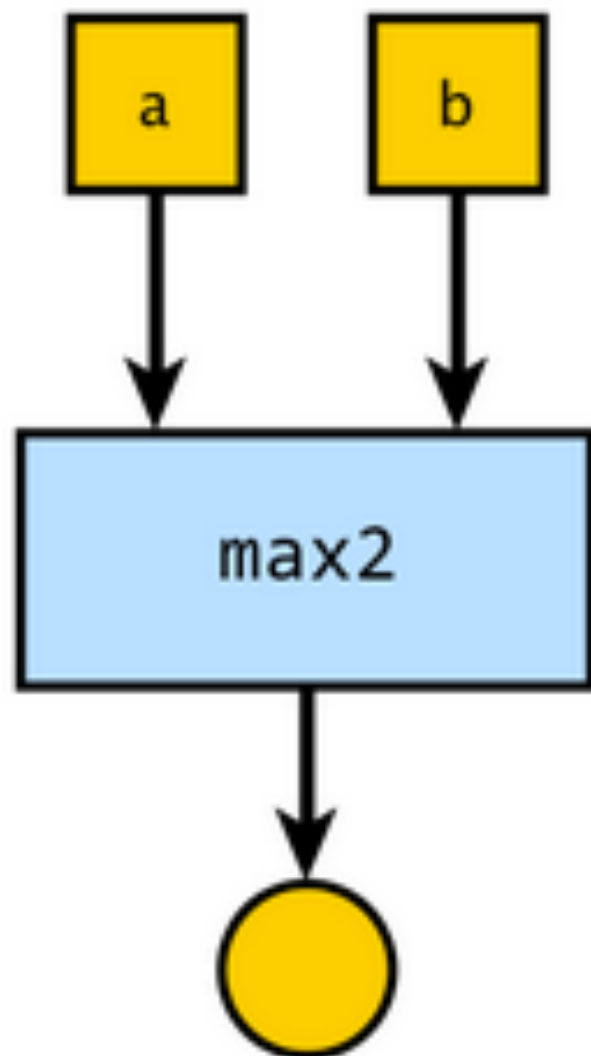
8) Resultatet som returneras är 14.

9) Funktionen double **anropas** med **argumentet** 3.5.

10) Resultatet som returneras är 7.0.

Funktionen max2() grafiskt

På bilden nedan ser vi en grafiskt representation av funktionen **max2()**. Funktionen tar två parametrar **a** och **b** och skall returnera det tal som är störst av dessa.



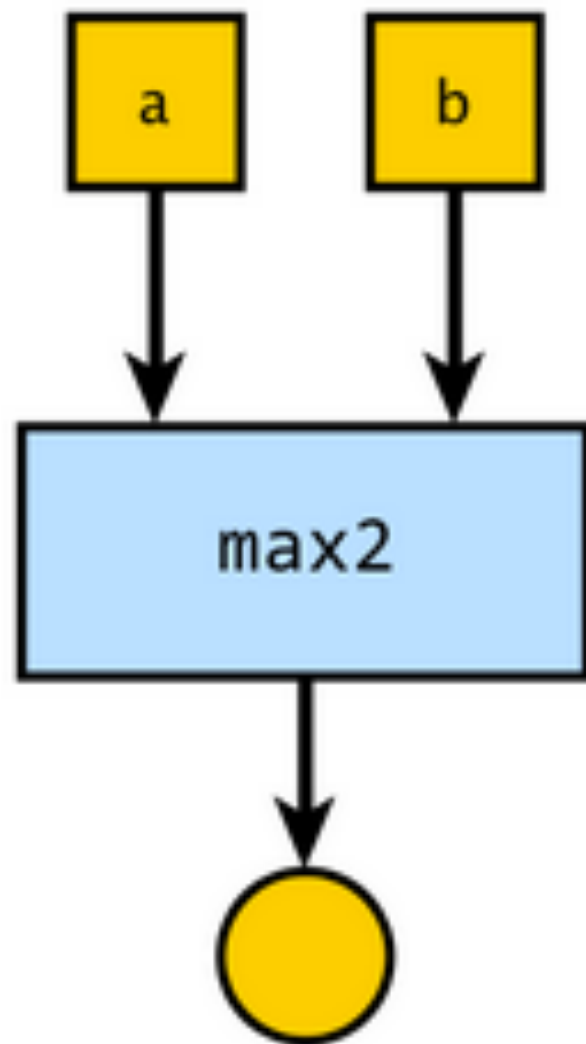
Parametrar

När vi anropar funktionen skall vi inte behöva känna till hur den bär sig åt för att beräkna resultatet.

Returvärde (resultat)

Funktionen max2() i Python

Grafiskt representation av funktionen max2().



Definition av funktionen max2() i Python.

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




```
>>> def max2(a, b):  
...     5 if a > b:  
...         6 return a  
...     7 else:  
...         8 return b  
>>>
```

- 1) När vi definierar en funktion i Python måste vi börja med **def**.
- 2) Namnet på funktionen, i detta fall **max2**.
- 3) Inom **parenteser** namnger vi de **parametrar** som funktionen beror av separerade av **kommatecken**, i detta fall a och b.
- 4) **Kolon**.

- 5) Resterande rader måste vara **indragna** (tryck tab). I detta fall utför funktionen en if-sats som i sin tur kräver ett kolon.
- 6) Kom ihåg att det som skall utföras om if-villkoret är sant måste vara indraget ytterligare ett steg (tryck tab). Använder **return** för att berätta vilket resultatet av funktionen skall vara.
- 7) Else-gren, vad skall utföras om if-villkoret inte är sant. Villkoret måste avslutas med kolon.
- 8) Använder **return** för att berätta vilket resultatet av funktionen skall vara.

Parametrar och argument.

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

Parameter

En funktion kan ta noll eller flera namngivna parametrar. Dessa parametrar kan sedan användas som variabler i funktionen.

Anropa funktion max2 i Python

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

9 >>> max2(2, 4)

10 4

11 >>> max2(22.75, 4)

12 22.75

- 9) Funktionen max2 **anropas** med **argumenten** 2 och 4.
- 10) Resultatet som returneras är 4.
- 11) Funktionen max2 **anropas** med **argumenten** 22.75 och 4.
- 12) Resultatet som returneras är 22.75.

Parametrar och argument.

```
>>> def max2(a, b):  
...     if a > b:  
...         return a  
...     else:  
...         return b  
...  
>>> max2(2, 4)  
4  
>>> max2(22, 4)  
22  
>>>
```

Parameter

En funktion kan ta noll eller flera namngivna parametrar. Dessa parametrar kan sedan användas som variabler i funktionen.

Argument

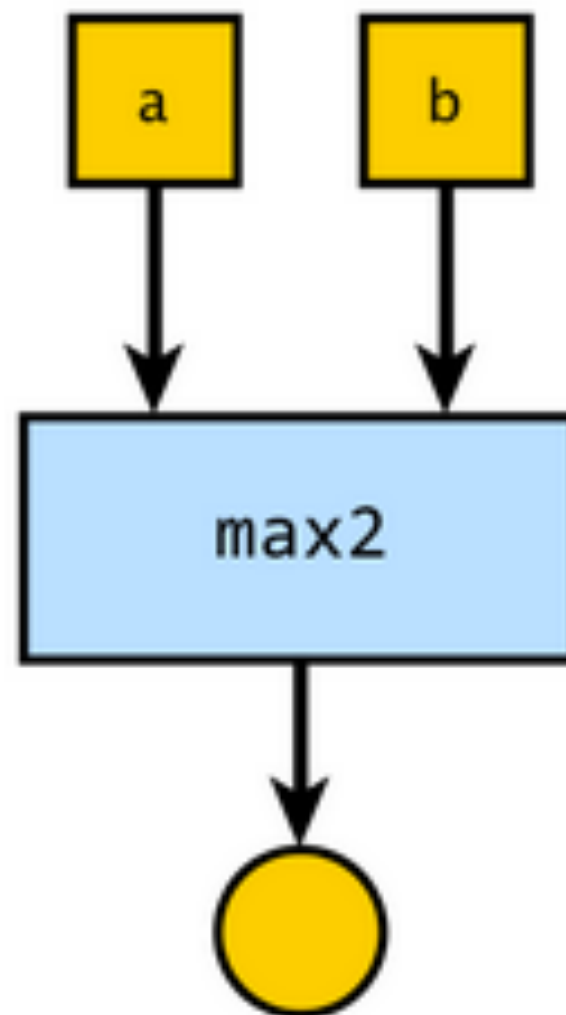
Vid funktionsanrop måste antalet argument överensstämma med antal parametrar. Vid anropet binds parametrarna till argumentens värden.

Söndra och härska

(1)

Med hjälp av funktioner kan vi bryta ner ett problem i mindre väl avgränsade delar för att sedan sätta samman delarna till en helhet.

Kan vi använda funktionen `max2()` som en del i lösningen av andra (större) problem?

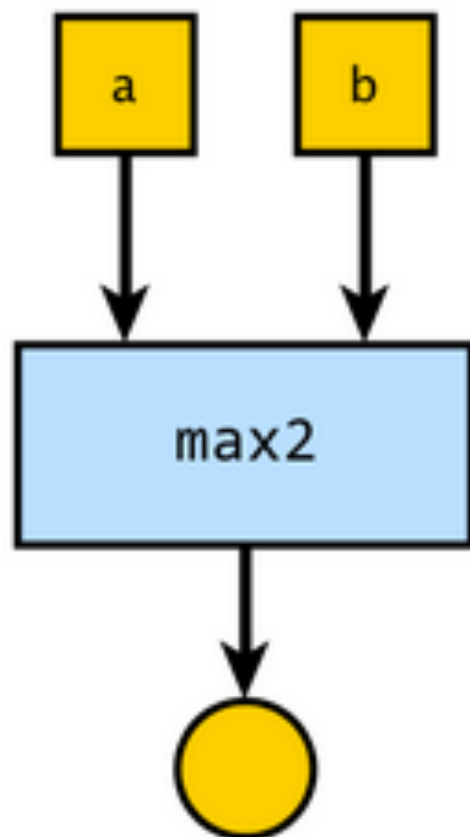


Söndra och härska

(2)

Med hjälp av funktioner kan vi bryta ner ett problem i mindre väl avgränsade delar för att sedan sätta samman delarna till en helhet.

Parametrar

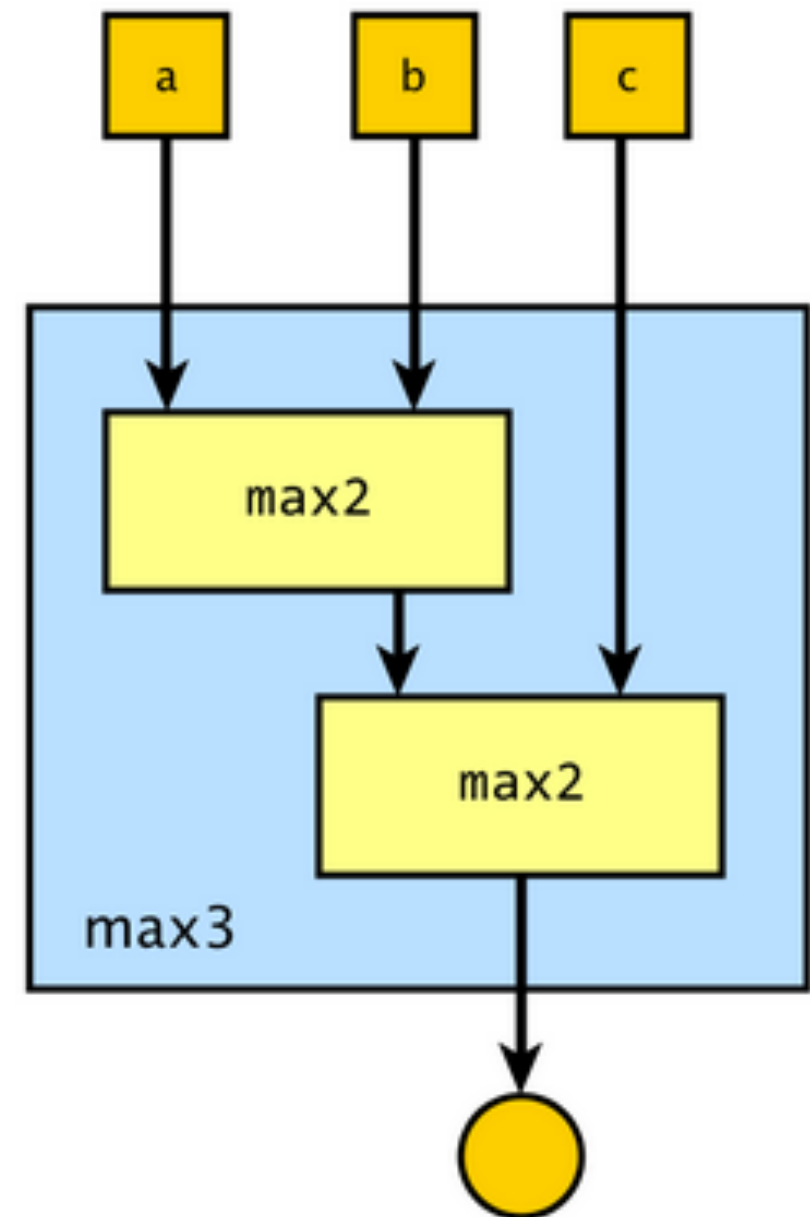


Resultat

Funktionen `max2()` med parametrar `a` och `b` ...

... kan användas för att bygga upp funktionen `max3()` med parametrar `a`, `b` och `c`.

Parametrar



Resultat

Utvärdering av **funktionsanrop** steg för steg

Utvärdering av **funktionsanrop** steg för steg

```
def double( n ):
    return n * 2
```

```
def tripple( n ):
    return 3 * n
```

```
x = double( 11 - tripple(3) )
```


Utvärdering av **funktionsanrop** steg för steg

```
def double( n ):
    return n * 2
```

```
def tripple( n = 3 ):
    return 3 * n
```

```
x = double( 11 - tripple(3) )
```

Utvärdering av **funktionsanrop** steg för steg

```
def double( n ):
    return n * 2
```

```
def tripple( n = 3 ):
    return 3 * n
```

```
x = double( 11 - tripple(3) )
```

Utvärdering av **funktionsanrop** steg för steg

```
def double( n ):
    return n * 2
```

```
def tripple( n = 3 ):
    return 3 * 3
```

```
x = double( 11 - tripple(3) )
```

Utvärdering av **funktionsanrop** steg för steg

```
def double( n ):
    return n * 2
```

```
def tripple( n = 3 ):
    return 3 * 3
```

```
x = double( 11 - tripple(3) )
```

Utvärdering av **funktionsanrop** steg för steg

```
def double( n ):
    return n * 2
```

```
def tripple( n = 3 ):
    return 9
```

```
x = double( 11 - tripple(3) )
```

Utvärdering av **funktionsanrop** steg för steg

```
def double( n ):
    return n * 2
```

```
def tripple( n ):
    return 3 * n
```

```
x = double( 11 - 9 )
```

Utvärdering av **funktionsanrop** steg för steg

```
def double( n ):
    return n * 2
```

```
def tripple( n ):
    return 3 * n
```

```
x = double( 11 - 9 )
```

Utvärdering av **funktionsanrop** steg för steg

```
def double( n ):
    return n * 2
```

```
def tripple( n ):
    return 3 * n
```

```
x = double( 2 )
```


Utvärdering av **funktionsanrop** steg för steg

```
def double( n ):
    return n * 2
```

```
def tripple( n ):
    return 3 * n
```

```
x = double( 2 )
```

Utvärdering av **funktionsanrop** steg för steg

```
def double( n = 2 ):
    return n * 2
```

```
def tripple( n ):
    return 3 * n
```

```
x = double( 2 )
```

Utvärdering av **funktionsanrop** steg för steg

```
def double( n = 2 ):
    return n * 2
```

```
def tripple( n ):
    return 3 * n
```

```
x = double( 2 )
```

Utvärdering av **funktionsanrop** steg för steg

```
def double( n = 2 ):
```

```
    return 2 * 2
```

```
def tripple( n ):
```

```
    return 3 * n
```

```
x = double( 2 )
```

Utvärdering av **funktionsanrop** steg för steg

```
def double( n = 2 ):
```

```
    return 2 * 2
```

```
def tripple( n ):
```

```
    return 3 * n
```

```
x = double( 2 )
```

Utvärdering av **funktionsanrop** steg för steg

```
def double( n = 2 ):
```

```
    return 4
```

```
def tripple( n ):
```

```
    return 3 * n
```

```
x = double( 2 )
```

Utvärdering av **funktionsanrop** steg för steg

```
def double( n ):
    return n * 2
```

```
def tripple( n ):
    return 3 * n
```

x = 4

www.python tutor.com

På denna sida finns ett mycket bra verktyg för att förstå hur Python fungerar.

```
>>> def max2(a, b):  
...     if a > b:  
...         return a  
...     else:  
...         return b  
...  
>>> max2(2, 4)  
4  
>>> max2(22, 4)  
22
```

En snabbänk med denna kod i Python Tutor.
Länken finns även i Studentportalen.

<http://tinyurl.com/nvbp8fw>

Lokala variabler

Lokala variabler skapas vid funktionsanrop och försvinner efter det att funktionen returnerat.

Skriv in och kör följande kod steg för steg i Python tutor.

```
def foo(x):  
    y = 2*x  
    return y + 1
```

```
print foo(11)
```

```
print y
```

Vad händer?

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	?

Python Tutor

<http://tinyurl.com/qxqbfqy>

Länken finns även på hemsidan om Problemlösning och programmering > Del 1 > Syfte och översikt > Workshop.

```
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.

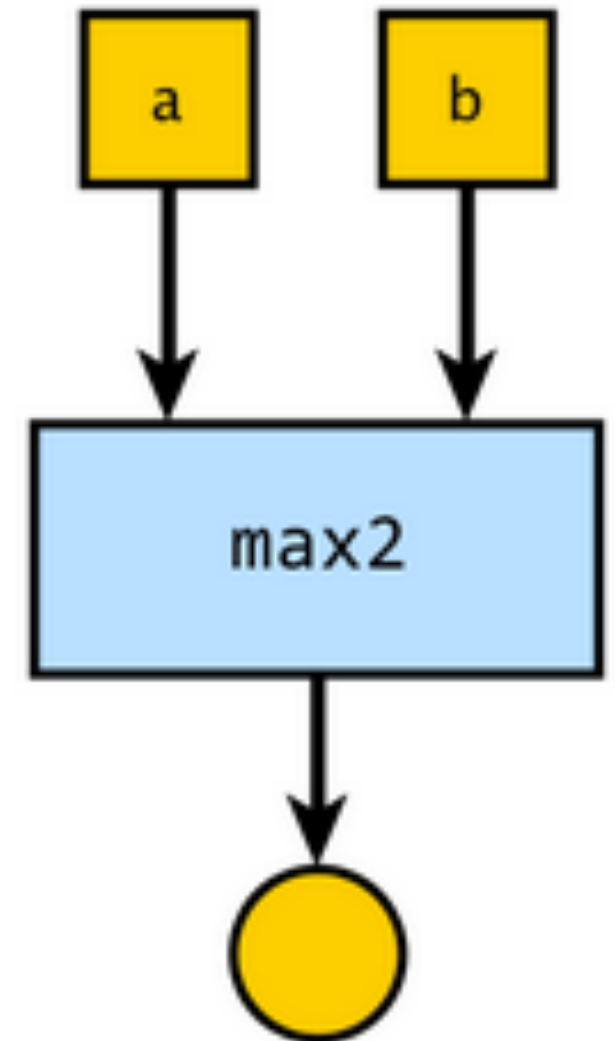
Det anses vara dålig programmeringssed att införa sidoeffekter eftersom de drabbar programvarans överskådlighet och dessutom ofta inför felaktigheter i programvarans funktion.

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 anropare 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.

Sidoeffekt

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

```
1 a = [1, 9, 3]
2
3 def foo(xs):
4     xs.append(55)
5
6     return 1*xs[0] + 2*xs[1] + 3*xs[2]
7
8 print a
9 z = foo(a)
10 print z
11 print a
```

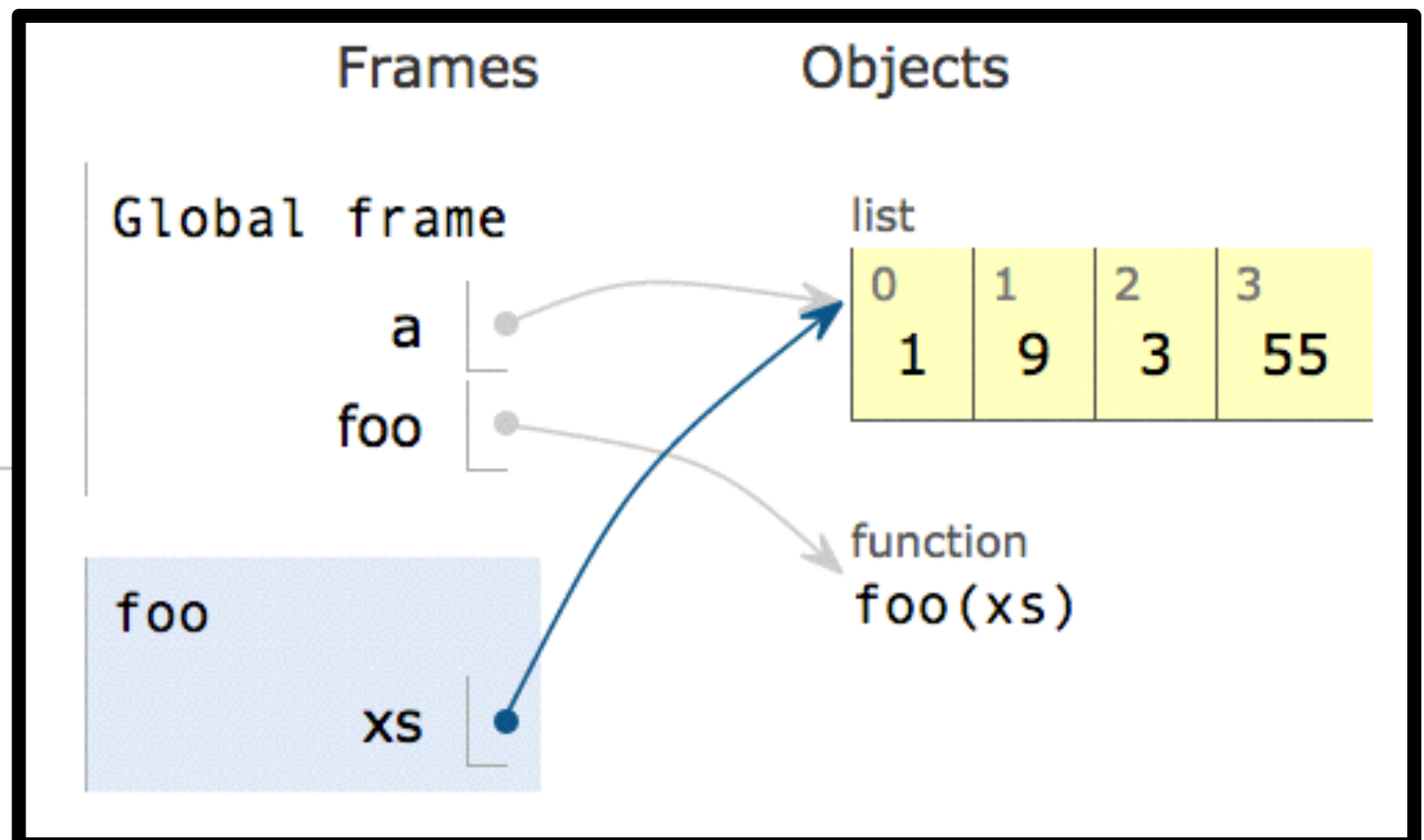
Utskrift när programmet kört klart?

Python Tutor

<http://tinyurl.com/pg325rr>

Länken finns även på hemsidan om Problemlösning och programmering > Del 1 > Syfte och översikt > Workshop.

```
1  a = [1, 9, 3]
2
3  def foo(xs):
4      xs.append(55)
5
6      return 1*xs[0] + 2*xs[1] + 3*xs[2]
7
8  print a
9  z = foo(a)
10 print z
11 print a
```



```
1 a = [1, 9, 3]
2
3 def foo(xs):
4     xs.append(55)
5
6     return 1*xs[0] + 2*xs[1] + 3*xs[2]
7
8 print a
9 z = foo(a)
10 print z
11 print a
```

Utskrift när programmet kört klart.

[1, 9, 3]

28

[1, 9, 3, 55]

```
1 a = [1, 9, 3]
2
3 def foo(xs):
4     xs.append(55)
5
6     return 1*xs[0] + 2*xs[1] + 3*xs[2]
7
8 print a
9 z = foo(a)
10 print z
11 print a
```

Utskrift när programmet kört klart.

[1, 9, 3]

28

[1, 9, 3, 55]

Funktionen **foo()** har sidoeffekten att listan som ges som argument inte är oförändrad efter anrop.

Spara Pythonkod i en fil (modul)

Spara följande kod i en fil med namn **test.py**.

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

Från Linux-teterminalen, navigera till den katalog där du sparade filen **test.py** och starta Python.

```
linux> ls  
max.py  
linux> python  
>>>
```

Importera modul

Läs in den sparade koden med hjälp av **import**.
Om filen heter test.py måste du skriva import test (dvs test utan .py).

```
>>> import test
>>> test.max2(2,4)
4
>>>
```

För att anropa funktion max2 i modulen test skriver vi modulnamnet (test) följt av punkt och sedan funktionsanropet på vanligt sätt.

Reload

Om du ändrar och sparar en modul du redan importerat måste du använda **reload** och inte import för att dina ändringar skall få effekt.

Notera att det är parenteser vid reload och utan parenteser vid import.

```
>>> import test
>>> x = test.max2(23,8)
>>> Ändrar och sparar modulen test.py
>>> reload(test)
>>> y = test.max2(23,8)
```