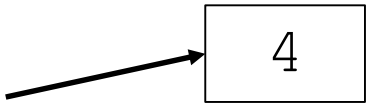
referenser

I Python har program åtkomst till värden via referenser.
En referens är ett namn på en specifik plats i minnet för ett värde.
När en variabel *tilldelas* ett värde, exvis:

```
x = 4
```

Allokeras en plats i minnet där värdet 4 lagras.



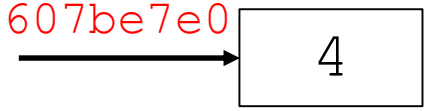
Eftersom 4 är ett heltal lagras det på heltalsvis i binärform

Adressen till denna minnesplats är egentligen ointressant. Det som är intressant är att vi refererar till minnesplatsen som `x` och att där är lagrat ett heltal. Variabeln `x` är därför av datatypen `int`. Vi kan ta reda på adressen mha den inbyggda funktionen *id*:

```
print('Adress x =', hex(id(x)))
```

```
Adress x = 0x607be7e0
```

`id(x)` ger adressen för `x` och `hex(id(x))` ger adressen utryckt hexadecimal.

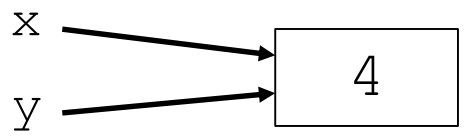


```
x = 4
y = x
print(x,y)
print('Adress x =', hex(id(x)), '. Adress y = ', hex(id(y)))
```

```
4 4
```

```
Adress x = 0x607be7e0 . Adress y =  0x607be7e0
```

`x` och `y` refererar samma minnesenhet



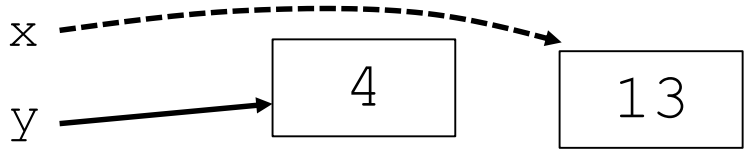

```
x += 9
```

Detta är en tilldelning av nytt värde, och nytt minne allokeras för `x`

```
print(x,y)
print('Adress x =', hex(id(x)), '. Adress y = ', hex(id(y)))
```

```
13 4
```

```
Adress x = 0x607be880 . Adress y =  0x607be7e0
```

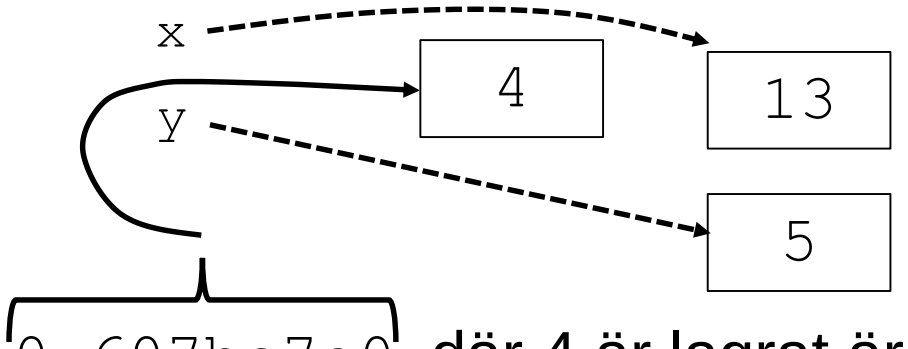


```
y = 5
```

Detta är en tilldelning av nytt värde, och nytt minne allokeras för `y`

```
print(x,y)
```
```
13 5
```

```
print('Adress x =', hex(id(x)), '. Adress y = ', hex(id(y)))
```
```
Adress x = 0x607be880 . Adress y =  0x607be7f0
```



Adressen `0x607be7e0`, där 4 är lagrat är ingen variabel refererad till.

```
z = 0
```

En loop som snurrar fem varv

```
for i in range(5):
    z = z+1
    print('z =', z, ' Adress z = ' , hex(id(z)))
```

```
z = 1  Adress z =  0x607be7b0
z = 2  Adress z =  0x607be7c0
z = 3  Adress z =  0x607be7d0
z = 4  Adress z =  0x607be7e0
z = 5  Adress z =  0x607be7f0
```

För varje tilldelning/uppdatering av värdet allokeras nytt minne. ”Gammalt” minnesinnehåll ”tappas bort”. Kan minnet ta slut?
Python har en mekanism som kallas *garbage collection* som tar hand om borttappat minne och återanvänder detta.

```
z = 0
```

En loop som snurrar fem varv

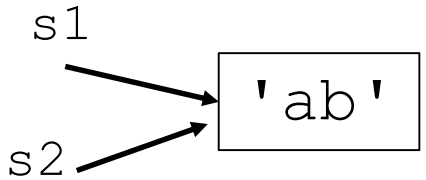
```
for i in range(5):
    z = z+1
    print('z =', z, ' Adress z = ' , hex(id(z)))
```

```
z = 1  Adress z =  0x607be7b0
z = 2  Adress z =  0x607be7c0
z = 3  Adress z =  0x607be7d0
z = 4  Adress z =  0x607be7e0
z = 5  Adress z =  0x607be7f0
```

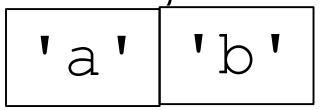
För varje tilldelning/uppdatering av värdet allokeras nytt minne. ”Gammalt” minnesinnehåll ”tappas bort”. Kan minnet ta slut?
Python har en mekanism som kallas *garbage collection* som tar hand om borttappat minne och återanvänder detta.

Strängar och referenser

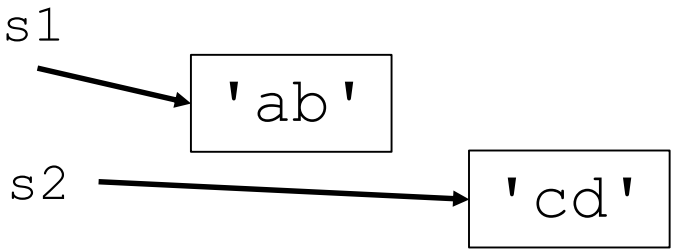
```
s1 = 'ab'
s2 = s1
```



Egentligen uppdelat i två enheter, två tecken

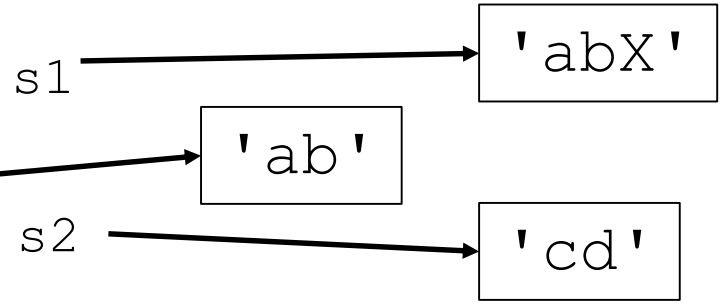



```
s2 = 'cd'
```




```
s1 = s1 + 'X'
```

Borttappat



Listor och referenser

```
v1 = [3,15,7]
v2 = [-4,6,3,12]
print('v1@ =', hex(id(v1)), '. v2@ = ', hex(id(v2)))
```

```
v1@ = 0x3147d68 . v2@ =  0x3147ce8
```

```
v2 = v1
print('v1@ =', hex(id(v1)), '. v2@ = ', hex(id(v2)))
```

```
v1@ = 0x3147d68 . v2@ =  0x3147d68
```

`v2` refererar till samma lista som `v1`

Tappat bort referensen till `[-4,6,3,12]`

```
v1.append(8)
print('v1@ =', hex(id(v1)))
```

```
v1@ = 0x2fe7d68
```

Referensen oförändrad

```
v1.remove(3)
print('v1@ =', hex(id(v1)))
```

```
v1@ = 0x2fe7d68
```

Referensen oförändrad

```
v3 = []
v4 = None
print('v3@ =', hex(id(v3)), '. v4@ = ', hex(id(v4)))
```

```
v3@ = 0x3150148 . v4@ =  0x607a8ce8
```

Studera följande exempel:

```
# x och n är parametrar (formella)
# parametrarna x och n samt variabeln y är lokala
def funcA(x, n):
    y = [x[0]]*n + x + [x[-1]]*n  # Utvidga lista
    return y

# Anrop
y = [1,2,3,4]
m = 2
z = funcA(y, m)  # y och m är argument (eller aktuella parametar)
print(z)  # [1, 1, 1, 2, 3, 4, 4, 4]
```

Vid anrop av funktion/metod kopieras argumentens värden till parametrarna.
Vad händer när programmet körs och vid anropet av funktionen?
Skall visualisera varje steg när programmet körs mha: www.pythontutor.com/

Vi gör följande förändringar:

```
def funcA(x, n):
    # x[0] = -1
    # n = 3
    y = [x[0]]*n + x + [x[-1]]*n  # Utvidga lista
    # x = [3,7]
    return y
```

Studerar vilka konsekvenser detta får för det tre fallen.