Funktioner

Programmering 1

24/25

Innehåll

Inbyggda funktioner

Exempel

I Python finns det en hel del inbyggda funktioner, exempelvis:

```
1 max(x)
2 min(x)
3 len(x)
4 print(x)
```

Funktioner

Exempel

```
1 import random
2 # Den här koden skapar en slumpad lista och skriver ut saker om
      den
3 min_lista = [random.randint(0,99) for i in range(10)]
4 print(min_lista)
5 print(f"Största värdet är {max(min_lista)} i listan")
6 print(f"Minsta värdet är {min(min_lista)} i listan")
7 print(f"Längden på listan är {len(min_lista)} element")
```

Argument

De flesta funktionerna tar emot variabler som *argument*, även kallat *parametrar*. En funktion kan ta emot flera argument, exempelvis:

```
1 range(start,slut,steg)
2 max(a,b,c,d,e,f)
```

Eller inga argument:

```
1 print()
```

Innehåll

Skapa egna funktioner

def

I Python kan du skapa dina egna funktioner så här:

```
1 def min_funktion():
2   # Här skriver du kod du vill ska hända
```

När du har skapat en funktion kan du anropa den själv, som om den vore inbyggd:

```
1 min_funktion()
```

Skapa egna funktioner

Argument

Vill du skapa funktionen så att den tar emot argument och gör något med dem kan du skriva så här:

```
1 def dubbla(x):
2   return 2*x
```

Och nu kan du anropa funktionen så här:

```
1 dubbla(5)
```

Skapa egna funktioner

return

Om du har en funktion där du vill att den ska göra något med värdet du skickar in och sen ger dig en möjlighet att spara det nya värdet så behöver du använda dig utav `return`:

```
1 def dubbla(x):
2   return 2*x
3
4 a = dubbla(5)
5 print(a)
```

Skapa egna funktioner

Exempel

En funktion kan också vara mycket mer komplicerad än de som visats i exemplen hittills:

```
1  def bubble_sort(lista):
2      for i in range(len(lista)): # Vi kommer att behöva gå
           igenom listan n gånger
3          for j in range(1,len(lista)-i): # Gå igenom varje tal
4              if lista[j-1] > lista[j]:  # Kolla om två grannar
                   är i fel ordning
5                  # Byt plats på talen
6                  temp = lista[j]
7                  lista[j] = lista[j-1]
8                  lista[j-1] = temp
9    return lista # Skickar tillbaka en sorterad lista
```

Innehåll

Global vs. Local

När man skapar en variabel inuti en funktion så existerar den bara inuti den funktionen.

```
1 def skriv_ut_tal_till_10():
2   x = 1
3   while x <= 10:
4     print(x)
5     x += 1
```

Nu finns variabeln `x` bara inuti funktionen `skriv_ut_tal_till_10`

Scope

Skulle vi däremot skapa en variabel utanför funktionen så kan man komma åt den i funktionen:

```
1 def min_funktion():
2   a = 3+x
3   return a
4
5 x = 5
6 print(min_funktion())
```

Scope

Vi kan däremot inte göra följande:

```
1 def min_andra_funktion():
2   x += 3
3   return x
4
5 x = 5
6 print(min_andra_funktion())
```

Kör vi den koden så får vi felmeddelandet `UnboundLocalError: local variable 'x'referenced before assignment`. Lite förenklat kan man säga att man får se men inte röra.

Scope

Dessutom så kan vi inte ändra på något som ligger utanför funktionen:

```
1 def foo():
2   x = 1
3   return x
4
5 x = 3
6 print("x:", foo()
7 print("x:", x)
```

Scope

Listor

Här är däremot listor ett stort undantag. Med listor kan man göra så som du inte kunde göra ovanför, och det behöver du akta dig för.

```
1 def bar(a):
2   a.append(3)
3
4 lista = [1,2]
5 bar(lista)
6 print(lista)
```

Innehåll

Övningar

1. Skriv en egen funktion som tar emot två tal och räknar ut talet mitt mellan de båda talen (medelvärdet)
2. Skriv en egen funktion som tar emot en lista med tal och räknar ut medelvärdet av talen.
3. Skriv en egen funktion som tar emot en textsträng och räknar ut hur många `'a'` det finns i den.
4. Utveckla funktionen så att den räknar ut hur många det finns av valfri bokstav.
5. Skriv en egen funktion som tar emot en lista med tal och skickar tillbaka det största talet i listan. (Använd inte `max()`)