

Föreläsning 3

Om funktioner och lite om debugging och kodstil

Vi testar att SPELA IN.

Frågor på rasterna från chatten ...

Strukturerera kod

```
82 def on_key_press(self, symbol, modifiers):
83     # Delegate key presses
84     if self.context_index == -1:
85         if symbol == key.UP and not self.active_index == 0:
86             self.menu_labels[self.active_index].color = [255, 255, 255, 255]
87             self.active_index -= 1
88             self.mags_dt = self.get_act_color_mag()
89         elif symbol == key.DOWN and not self.active_index == 3:
90             self.menu_labels[self.active_index].color = [255, 255, 255, 255]
91             self.active_index += 1
92     self.mags_dt = self.get_act_color_mag()
93
```

```
93 import tokenize #java.util.*;
94 import io, math #import java.io.*;
95 import myParser
96
97
98
99 class Calculator: #klassen ska börja med stor bokstav
100     def __init__(self, token, v):
101         self.token = token
102         self.variables = v #{"PI": math.pi, "E": math.e, "ans": 0.0}
103         self.parser = parser.Parser(self.token, self.variables)
104
105     def statement():
106         self.token_type, self.token_string, _, self.end, _ = next(token)
107         print('i statement.', self.token_string)
108         while self.end == tokenizer.NEWLINE:
109             self.token_type, self.token_string, _, self.end, _ = next(token)
110             print('inte newline')
111         command = self.token_string
112         print(command)
113
114
115 print("Numerical calculator version 2013-04-10")
116 s=input("> ")
117 code_buffer = io.StringIO(s)
118 token = tokenize.generate_tokens(code_buffer.readline)
119 variables = {"PI": math.pi, "E": math.e, "ans": 0.0}
120 p = myParser.My_parser(token, variables)
121 print('h')
122 calc = Calculator(token, variables)
123 while (true):
124     calc.statement()
```

```
def term(self):
    prod = self.factor()
    #self.token_type, self.token_string, _, _, _ = next(token)
    while (self.token_string == '*' or self.token_string == '/'):
        if self.token_string == '*':
            self.token_type, self.token_string, _, _, _ = next(token)
            prod = prod * self.factor()
            print('i term prod=', prod)
        if self.token_string == '/':
            self.token_type, self.token_string, _, _, _ = next(token)
            namnare = self.factor() #int(self.token_string)
            if namnare == 0.: # EXCEPTION HÄR!
                print('Division by 0')
                return -1
            prod = prod/namnare
    return prod

def factor(self):
    return self.primary()

def primary(self):
    result = 99999
    print('primary self.token_type', self.token_type)
    if self.token_string == '(':
        self.token_type, self.token_string, _, _, _ = next(token)
        result = self.assignment()
    if self.token_string == ')':
        self.token_type, self.token_string, _, _, _ = next(token)
        return result
    else:
        print("Expected '(')") # SYNTAX-EXCEPTION HÄR!

elif self.token_type == 2: # tokenize.NUMBER:
    print('i primary, elif type==2', self.token_string)
    result = float(self.token_string)
    self.token_type, self.token_string, _, _, _ = next(token)
    return result

elif self.token_string == '-': #negering token_type == 54
    self.token_type, self.token_string, _, _, _ = next(token)
    result = -self.primary()
    -----
```

Programvara är ofta stor och komplex. 100-tals programmerare kan arbeta med olika delar av koden. Det behövs en struktur. Abstraktion.

Strukturerera kod

```
82 self.color = colors
83
84 def on_key_press(self, symbol, modifiers):
85     # Update the cursor
86     if self.context_index == -1:
87         if symbol == key.UP and not self.active_index == 0:
88             self.menu_labels[self.active_index].color = [255, 255, 255, 255]
89             self.active_index -= 1
90             self.mags_dt = self.get_act_color_mag()
91         elif symbol == key.DOWN and not self.active_index == 3:
92             self.menu_labels[self.active_index].color = [255, 255, 255, 255]
93             self.active_index += 1
94         else:
95             return
96
97 class Calculator: #klassen ska börja med stor bokstav
98     def __init__(self, token, v):
99         self.token = token
100         self.variables = v #{"PI": math.pi, "E": math.e, "ans": 0.0}
101         self.parser = parser.Parser(self.token, self.variables)
102
103     def statement():
104         self.token_type, self.token_string, _, self.end, _ = next(token)
105         print('i statement.', self.token_string)
106         while self.end == tokenizer.NEWLINE:
107             self.token_type, self.token_string, _, self.end, _ = next(token)
108             print('inte newline')
109         command = self.token_string
110         print(command)
```

```
import tokenize #java.util.*;
import io, math #import java.io.*;
import myParser

class Calculator: #klassen ska börja med stor bokstav
    def __init__(self, token, v):
        self.token = token
        self.variables = v #{"PI": math.pi, "E": math.e, "ans": 0.0}
        self.parser = parser.Parser(self.token, self.variables)

    def statement():
        self.token_type, self.token_string, _, self.end, _ = next(token)
        print('i statement.', self.token_string)
        while self.end == tokenizer.NEWLINE:
            self.token_type, self.token_string, _, self.end, _ = next(token)
            print('inte newline')
        command = self.token_string
        print(command)

print("Numerical calculator version 2013-04-10")
s=input("> ")
code_buffer = io.StringIO(s)
token = tokenize.generate_tokens(code_buffer.readline)
variables = {"PI": math.pi, "E": math.e, "ans": 0.0}
p = myParser.My_parser(token, variables)
print('h')
calc = Calculator(token, variables)
while (true):
    calc.statement()
```

```
def term(self):
    prod = self.factor()
    #self.token_type, self.token_string, _, self.end, _ = next(token)
    while (self.token_string == '*' or self.token_string == '/'):
        if self.token_string == '*':
            self.token_type, self.token_string, _, self.end, _ = next(token)
            prod = prod * self.factor()
            print('i term prod=', prod)
        if self.token_string == '/':
            self.token_type, self.token_string, _, self.end, _ = next(token)
            namnare = self.factor() #int(self.token_string)
            if namnare == 0.: # EXCEPTION HÄR!
                print('Division by 0')
                return -1
            prod = prod/namnare
    return prod

def factor(self):
    return self.primary()

def primary(self):
    result = 99999
    print('primary self.token_type', self.token_type)
    if self.token_string == '(':
        self.token_type, self.token_string, _, self.end, _ = next(token)
        result = self.assignment()
        if self.token_string == ')':
            self.token_type, self.token_string, _, self.end, _ = next(token)
            return result
    else:
        print("Expected '(')") # SYNTAX-EXCEPTION HÄR!

    elif self.token_type == 2: # tokenize.NUMBER:
        print('i primary, elif type=2', self.token_string)
        result = float(self.token_string)
        self.token_type, self.token_string, _, self.end, _ = next(token)
        return result

    elif self.token_string == '-': #negering token_type == 54
        self.token_type, self.token_string, _, self.end, _ = next(token)
        result = -self.primary()
        return result
```

En grundläggande strategi är att dela upp koden i **moduler**, filer, som delas in i **funktioner** som gör väl avgränsade uppgifter. Moduler kan läggas i paket.

Strukturerera kod

```
82 self.color = colors[...].mags.dt[i]
83
84 def on_key_press(self, symbol, modifiers):
85     # delegate key press
86     if self.context_index == -1:
87         if symbol == key.UP and not self.active_index == 0:
88             self.menu_labels[self.active_index].color = [255, 255, 255, 255]
89             self.active_index -= 1
90             self.mags.dt = self.get_act_color_mags()
91         elif symbol == key.DOWN and not self.active_index == 3:
92             self.menu_labels[self.active_index].color = [255, 255, 255, 255]
93             self.active_index += 1
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
```

```
import tokenize #java.util.*;
import io, math #import java.io.*;
import myParser

class Calculator: #klassen ska börja med stor bokstav
    def __init__(self, token, v):
        self.token = token
        self.variables = v #{"PI": math.pi, "E": math.e, "ans": 0.0}
        self.parser = parser.Parser(self.token, self.variables)

    def statement():
        self.token_type, self.token_string, _, self.end, _ = next(token)
        print('i statement.', self.token_string)
        while self.end == tokenizer.NEWLINE:
            self.token_type, self.token_string, _, self.end, _ = next(token)
            print('inte newline')
        command = self.token_string
        print(command)

print("Numerical calculator version 2013-04-10")
s=input("> ")
code_buffer = io.StringIO(s)
token = tokenize.generate_tokens(code_buffer.readline)
variables = {"PI": math.pi, "E": math.e, "ans": 0.0}
p = myParser.My_parser(token, variables)
print('h')
calc = Calculator(token, variables)
while (true):
    calc.statement()
```

```
def term(self):
    prod = self.factor()
    #self.token_type, self.token_string, _, self.end, _ = next(token)
    while (self.token_string == '*' or self.token_string == '/'):
        if self.token_string == '*':
            self.token_type, self.token_string, _, self.end, _ = next(token)
            prod = prod * self.factor()
            print('i term prod=', prod)
        if self.token_string == '/':
            self.token_type, self.token_string, _, self.end, _ = next(token)
            namnare = self.factor() #int(self.token_string)
            if namnare == 0.: # EXCEPTION HÄR!
                print('Division by 0')
                return -1
            prod = prod/namnare
    return prod

def factor(self):
    return self.primary()

def primary(self):
    result = 99999
    print('primary self.token_type', self.token_type)
    if self.token_string == '(':
        self.token_type, self.token_string, _, self.end, _ = next(token)
        result = self.assignment()
        if self.token_string == ')':
            self.token_type, self.token_string, _, self.end, _ = next(token)
            return result
    else:
        print("Expected ')'" ) # SYNTAX-EXCEPTION HÄR!

    elif self.token_type == 2: # tokenize.NUMBER:
        print('i primary, elif type=2', self.token_string)
        result = float(self.token_string)
        self.token_type, self.token_string, _, self.end, _ = next(token)
        return result

    elif self.token_string == '-': #negering token_type == 54
        self.token_type, self.token_string, _, self.end, _ = next(token)
        result = -self.primary()
        return result
```

Många färdiga moduler som ofta används. Ex: math: beräkna sinus av en vinkel eller beräkna roten ur ett tal... Ex: rita på skärmen, ...

Funktioner och metoder i Python

F3: Strukturera kod mha funktioner (det finns fler sätt att strukturera kod på)

- Hittills: skrivit kod som exekveras sekventiellt.
- Långt, upprepningar?

Funktioner:

- Dela in koden i "delar" med specifika uppgifter
- Lättare att läsa!
- Lättare att felsöka
- Lättare att återanvända (ex sin-funktionen finns redan)
- Undvik upprepning av kod
- funktionerna gör typiskt *en* uppgift och är lättare att återanvända än ett helt program

Funktioner och metoder i Python

Exempel hittills: Funktioner känns (ofta) igen på parenteser:

```
namn = input('Vad heter du?')
```

```
hyp = math.sqrt(k1*k1 + k2*k2)
```

Funktioner som följer med Python

Inbyggda

```
print()
```

```
input()
```

```
int()
```

```
...
```

importera

```
math.sin()
```

```
turtle.forward()
```

```
csv.reader()
```

```
...
```

Vanliga funktioner, behöver ofta laddas ned

```
matplotlib.pyplot()
```

```
matplotlib.xscale()
```

```
...
```

Funktioner du/ni skriver själv

Lektion 4 --

[Python's built-in functions: https://docs.python.org/3/library/functions.html](https://docs.python.org/3/library/functions.html)

Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

Built-in Functions				
<code>abs()</code>	<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>
<code>all()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>any()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>ascii()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>bin()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bool()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>breakpoint()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	

abs(x)

Return the absolute value of a number. The argument may be an integer, a floating point number, or an object implementing `__abs__()`. If the argument is a complex number, its magnitude is returned.

all(iterable)

Exempel 1: Funktionen sinus

Matematik:

$$\sin(\pi/6) = 0.5$$

Python:

`sin(pi/6)`

`#Kompileringsfel!`

Exempel 1: Funktionen sinus

Matematik:

$$\sin(\pi/6) = 0.5$$

Python:

```
import math
```

```
math.sin(math.pi/6)
```

```
#Vad är "problemet" med den här koden?
```

Exempel 1: Funktionen sinus

Matematik:

$$x = \sin(\pi/6)$$

ELLER:

$$\sin(\pi/6) = x$$

”x är lika med sinus $\pi/6$ radianer”

Python:

```
import math
```

```
x = math.sin(math.pi/6)
```

```
print(x)
```

```
math.sin(math.pi/6) = x # FEL!
```

(= *tilldelningsoperatorn*)

”x tilldelas det funktionen sin
returnerar för *argumentet* $\pi/6$
radianer”

Exempel 2: Funktionen har namnet **f**

Matematik:

$f(3)=20$ "f av 3 är 20"

Rätt eller fel beroende på hur funktionen ser ut

Python:

```
y = f(3) # "variabeln y tilldelas värdet av  
        # funktionsanropet med argumentet 3"  
print(y)  
# eller print(f(3))
```

Fungerar bara om vi definierat en funktion f (korrekt)

Exempel 2: Funktionen har namnet **f**

Matematik:

Funktionen:

$$f(x) = x^2 + 5x - 4$$

$$f(3) = 20$$

Python:

Funktionen definieras:

```
def f(x): # funktionshuvud
    y = x*x + 5*x - 4
    return y
```

```
y = f(3) # "y tilldelas värdet av
         # funktionsanropet f(3)"
print(y) #skriver ut 20
```

Exempel 3: Funktionen cirkelns area

Matematik:

Exempel:

Arean av en cirkel med radien 8:

$$a(8) \approx 200,96$$

Python:

Exempel:

Arean av en cirkel med radien 8:

```
a = c_area(8)
print(a)
```

Exempel 3: Funktionen cirkelns area

Matematik:

Funktionen för area av en cirkel:

$$a(r) = \pi r^2$$

*Parameter
(formell
Parameter)*

Python:

Funktionsdefinition för cirkelarea:

```
def c_area(r):  
    a = math.pi*r*r  
    return a # värdet returneras
```

Exempel: en cirkel med radien 8:

$$a(8) \approx 200,96$$

Exempel: en cirkel med radien 8:

```
a = c_area(8) #funktionen c_area anropas  
# variabeln a tilldelas värdet som returneras  
print(a)  
print(c_area(5.3))
```

*Argument
(aktuell
Parameter)*

Funktioner: programexekveringen

```
# först import-sater  
import math
```

```
# sen funktionsdefinitioner  
def c_area(radie):  
    a = math.pi*radie*radie  
    return a
```

1.

```
def rätblocksvolym(a, b, c):  
    return a*b*c
```

```
#Sen funktionsanrop mm:
```

```
area = c_area(5)
```

```
print('Cirkelns area är ', area)
```

```
print('Ett rätblock med måtten 2,3,8 har arean', rätblocksvolym(2,3,8))
```

```
print('Ett rätblock med måtten 7.1, 3.5 och 9 har arean', rätblocksvolym(7.1, 3.5, 9))
```

Programexekeringen

```
import math
```

```
def c_area(radie): # radie sätts till 5  
    a = math.pi*radie*radie  
    return a
```

```
def rätblockarea(a, b, c):  
    return a*b*c
```

```
#Funktionsanrop:      2.
```

```
area = c_area(5)
```

```
print('Cirkelns area är ', area)
```

```
print('Ett rätblock med måtten 2,3,8 har arean', rätblockarea(2,3,8))
```

```
print('Dubbla volymen är', 2*rätblocksvolym(2,3,8))
```


Programexekeringen

```
import math
```

```
def c_area(radie):
```

3. ↓ `a = math.pi*radie*radie`
↓ `return a`

```
def rätblockarea(a, b, c):
```

```
    return a*b*c
```

```
#Funktionsanrop:
```

```
area = c_area(5)
```

```
print('Cirkelns area är ', area)
```

```
print('Ett rätblock med måtten 2,3,8 har arean', rätblockarea(2,3,8))
```

```
print('Ett rätblock med måtten 7.1, 3.5 och 8.9 har arean', rätblockarea(7.1, 3.5, 9))
```

← Vad händer efter satsen `return a`?

1. Programmet kör funktionen `rätblockarea`
2. Programmet hoppar tillbaka till `area=c_area(5)`
3. Programmet hoppa till satsen under:
`print('Cirkelns area är', area)`

Programexekeringen

```
1 import math
```

```
2
```

```
3 def c_area(radie):
```

```
4     a = math.pi*radie*radie
```

```
5     return a
```

```
6
```

```
7 def rätblockarea(a, b, c):
```

```
8     return a*b*c
```

```
9
```

```
10 #Funktionsanrop:
```

```
11 area = c_area(5) # variabeln area heter inte likadant som den returnerande variabeln a
```

```
12 print('Cirkelns area är ', area)
```

```
13 print('Ett rätblock med måtten 2,3,8 har arean', rätblockarea(2,3,8)) # funktionen har flera parametrar
```

```
14 print('Ett rätblock med måtten 7.1, 3.5 och 8.9 har arean', rätblockarea(7.1, 3.5, 9)) #återanvänd
```

**4. VÄRDET returneras (inte bokstaven a),
ges till variabeln area**

- 1. Vilken av följande rader kan `a=c_area(5)` INTE flyttas till: rad 2, 6, 9 eller 15?**
- 2. Varför inte?**
- 3. Vilken/vilka rad(er) skulle fungera men är opassande?**

Ha koll på programmet (debugga)

Mitt program ger felmeddelande eller ger fel svar!

1. Läs din kod i exekverings-ordning.
2. Lägg in print-satser: `print('I funktionen calc. x=', x)`
3. Kommentera bort kod om du är osäker på var felet faktiskt är. Felmeddelanden kan peka på fel rad.

Edit -> Comment out (Alt+3 sen Alt+4)

MEN även:

Ha koll på programmet (debugga)

1. **View -> Variables** (högra övre fönstret flik Variables)
2. **DEBUGGERN: Ctrl + F5** Eller **Run -> Debug current script** Eller **'Iusen'**

Sen **F6** (snabb) eller **F7** (långsam) körning (F8 kör som vanligt)


Eller: Pilarna i meny-raden bredvid kör-pilen

Starta debugging inne i programmet:

Debug -> ställ markören på en rad -> 'run to cursor' ELLER

Dubbelklicka på radnummer=> brytpunkt. Kör deuggern

Se även Hjälp-fliken.

 Thonny - C:\Undervisning\Prog1_Python\F3\F3_ex2.py @ 5 : 16

File Edit View Run Device Tools Help



Funktioner utan return-sats (returnerar *None*)

```
def timmar(år): #Testa att byta return mot print
    return år*365*24
```

```
def namnsdag(namn): #Ingen return-sats. None returneras 'tyst'
    if namn == 'Harry':
        return
    print('Grattis på namnsdagen ', namn)
```

```
y = int(input('Hur gammal är du? '))
t = timmar(y)
print(f'Du är {y} år och har levt ungefär {t:.2f} timmar')
print('och ', timmar(y*60*60), ' sekunder\n')
```

```
namnsdag('Harry')
```

```
x = namnsdag('Anna')
print(x)    #Vad skrivs ut?
```

Funktioner kan ha 0, 1, ... parametrar. **Defaultparametrar**

```
def namnsdag(antal, namn='Anna'):  
    print('Grattis på namnsdagen ', namn)  
    print('HURRA!'*antal)
```

```
def lov(): #En funktion utan parametrar. Ingen info utifrån behövs.  
    print('Idag är det skollov.')  
    print('Välkomna tillbaka nästa vecka.')
```

```
#Anropa procedurerna:  
n = input('Vem har namnsdag idag?')  
ant = int(input('Hur många gånger vill du hurra?'))  
namnsdag(ant,n)  
namnsdag(ant) #Vad händer här?  
lov()  
#Fråga: hur många funktionsanrop innehåller koden?
```

Om exekveringen går fel...

```
def namnsdag(antal, namn='Anna'):
    if antal < 0:
        raise ValueError()
    print('Grattis på namnsdagen', namn)
    for i in range(antal):
        print('HURRA', namn, '!')

#Funktionsanrop:
n = input('Vem har namnsdag idag? ')
while True:
    a = int(input('Hur många gånger vill du hurra? '))
    try:
        namnsdag(a,n)
        break # Hoppa ur while-loopen
    except ValueError:
        print('\n*** Negativt argument till namnsdag!')
        print('Try again!')
```

Funktioner kan ha flera defaultparametrar

De måste ligga sist i parameterlistan.

```
def namnsdag(antal, förnamn='Anna', efternamn='Eckerdal'):  
    print('Grattis på namnsdagen ', förnamn, efternamn)  
    print('HURRA! '*antal)
```

#Anropa proceduren:

```
namnsdag(2)
```

```
namnsdag(3, 'Eva')
```

```
namnsdag(4, 'Bert', 'Klen')
```


Funktioner som parametrar (se lektion 7)

Funktioner kan skickas som argument till andra funktioner (Funktioner är objekt)

```
def shout(text):  
    return text.upper()
```

```
def whisper(text):  
    return text.lower()
```

```
def greet(func): # variabeln refererar till funktionen  
    greeting = func("Hi, I am created by a function passed as an argument.")  
    print(greeting)
```

```
print(shout('HÖR du mej?'))  
print(whisper('Skrik INTE!'))
```

```
greet(shout) # func i funktionen greet refererar till shout  
greet(whisper) # func i funktionen greet refererar till wisper
```

```
yell = shout # tilldelar funktionen till en variabel. Anropar inte funktionen  
print(yell('Hello')) # anropar yell, dvs shout  
silence = whisper  
print(silence('HI there!'))
```

Funktioner: Scope och lokala variabler

```
1 month = 0 #Fult!Global. Scope (räckvidd): resten av koden
2. def month_lived():
3     # Lokal variabel month överskuggar den globala, men lever bara inne i funktionen:
4     month = age*12
5     return month
6
7 print('month=', month)
8 age = 21 # Global
9 m = month_lived()
10 print('Du har levt c:a ', m, ' månader')
11
12 print('month=', month)
# Fråga: vd blir utskriften från rad 7 och 12?
```

Scope

```
def first_func(x): # Parametern x är en lokal variabel
    y=1 # y är en lokal variabel som finns bara i funktionen
    x = y+3 # Den lokala variabelen x ändras
    return x
```

```
def second_func():
    global x # Global variabel räcker utanför funktionen. Undvik helst.
    x = 3
    text = 'hejsan, '
    return text*x
```

```
x = 2
print('first_func: ', first_func(x)) # Det är värdet av x, dvs 2 som skickas till funktionen
print('x =',x)
# print(y) FEL. Y bara definierad i sitt scope, dvs i funktionen first_func
print('second_func: ', second_func())
print('x =',x)
```

Snygg, lättläst kod som följer konventionen.

```
import ...
```

```
def hypotenuse_rectangle():
```

```
....
```

```
def interest_per_month():
```

```
....
```

```
#code here:
```

```
variables
```

```
function calls...
```

```
X1=... # bad variable name
```

```
X2=...
```

```
X3=...
```

1. import-satser
2. Funktionsdefinitionerna
3. Resten av koden.

Två tomma rader mellan funktionsdefinitionerna.

Engelska, helst.

Liten bokstav i namn, _ för tydlighet. "Talande" namn.

OBS! Lägg inte variabler eller funktionsanrop innan eller mellan funktionsdefinitionerna.

PEP 8 -- Style Guide for Python Code

Se Minilektionen “**Kodningsregler**” på kurshemsidan:

- Limit all lines to a maximum of 79 characters. May use \
- Surround top-level function and class definitions with two blank lines.
- Method definitions inside a class are surrounded by a single blank line.
- A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition. Such a docstring becomes the `__doc__` special attribute of that object.
- For consistency, always use `"""triple double quotes"""` around docstrings.
- Write docstrings for all public modules, functions, classes, and methods.
- **Function and method names should be lowercase, with words separated by underscores as necessary to improve readability. (`my_function()`):)**
- **Variable and instance variable names follow the same convention as function names.**

Funktioner och metoder i Python

- Delar in koden i "delar"
- Undvik upprepning av kod: funktionerna gör typiskt en uppgift och är lättare att återanvända än ett helt program, ex `math.sin(v)`
- Lättare att testa och läsa
- Följ namnkonventionen. Bra funktions- och variabelnamn.
- **Funktions- och metदानrop** känns (oftast) igen på **parenteserna**
- **Funktionsdefinitioner:**
 - def** funktionsnamn(ev parametrar):
- Funktionsdefinitioner: läggs före anropen
- Funktioner utan return-sats returnerar **None**
- Först import-satse, sen funktionsdefinitioner, sen huvudprogrammet
- Lägg in tomma rader så programmet blir lättläst
- (Metoder: objekt.metodanrop() Punktnotation)

Swap! (Byt värde på två variabler)

```
#Swap!
```

```
a = 7
```

```
b = 5
```

```
print('Byt värde!')
```

```
a=b
```

```
b=a
```

```
print('a=',a, ' ' 'b=',b)
```

#DISKUTERA: 1. Vad skrivs ut? (testkör) 2. Rätta koden.

Funktionen swap (byt värden)

```
def swap(a, b): # a blir en kopia av x, b av y
```

```
    temp = a # spara a i temp
```

```
    a = b    # kopiera b till a
```

```
    b = temp # kopiera temp till b
```

```
x = 7
```

```
y = 5
```

```
print('x=',x, ' ' 'y=',y)
```

#DISKUTERA: 1. Vad skrivs ut? (testkör) Förklara.

Funktionen swap (byt värden)

```
def swap(a, b): # a blir en kopia av x, b av y
    temp = a # spara a i temp
    a = b    # kopiera b till a
    b = temp # kopiera temp till b
    return a,b # returnerar en tuple. Går även att returnera en lista
```

```
x = 7
y = 5
tup = swap(x,y) # x ska bli 5 och y 7
x = tup[0] ; y = tup[1]
print('Swaped: x=',x, ' ' 'y=',y)
```