Föreläsning 3

Om funktioner och lite om debugging och kodstil.

Föreläsningen anpassad för nybörjare!

Vi spelar in.

Frågor efteråt från chatten …

Strukturera kod

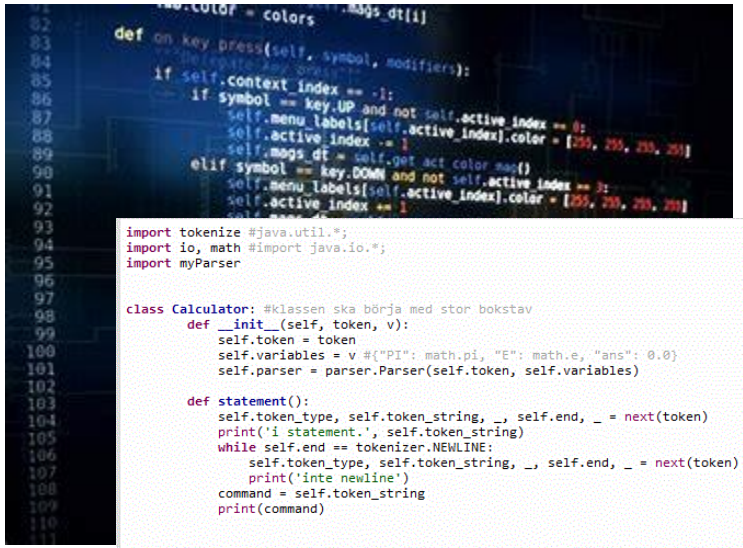
```
def on_key_press(self, symbol, modifiers):
    if self.context_index == -1:
        if symbol == key.UP and not self.active_index == 0:
            self.menu_labels[self.active_index].color = [255, 255, 255, 255]
            self.active_index -= 1
            self.mogs_dt = self.get_act_color_map()
        elif symbol == key.DOWN and not self.active_index == 3:
            self.menu_labels[self.active_index].color = [255, 255, 255, 255]
            self.active_index += 1
```

```
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, _, _, _ = 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.

Strukturera kod



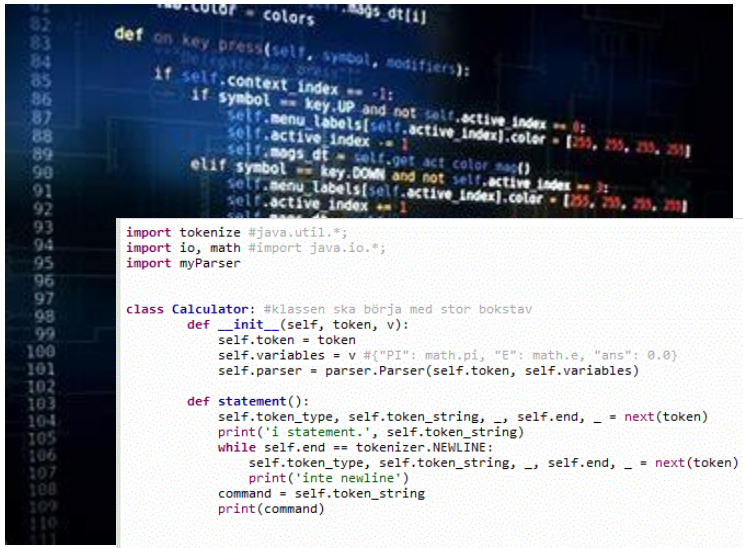
```
def on_key_press(self, symbol, modifiers):
    if self.context_index == -1:
        if symbol == key.UP and not self.active_index == 0:
            self.menu_labels[self.active_index].color = [255, 255, 255, 255]
            self.active_index -= 1
            self.msgs_dt = self.get_act_color_msg()
        elif symbol == key.DOWN and not self.active_index == 3:
            self.menu_labels[self.active_index].color = [255, 255, 255, 255]
            self.active_index += 1
```

```
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, _, _, _ = 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()
```

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

Strukturera kod



```
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, _, _, _ = 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()
```

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?

Varför 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/3.10/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		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

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*)

Skriv egna funktioner. 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

<u>Matematik:</u>

x = sin(π/6)

ELLER:

sin(π/6) = x

”x är lika med sinus π/6 radianer”

<u>Python:</u>

```
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* π/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):
    y=x*x + 5*x - 4
    return y

# Anropa funktionen f:
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:

```
ar = a(8)
print(ar)
```

Exempel 3: Funktionen cirkelns area

Matematik:

Funktionen för area av en cirkel:

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

Exempel: en cirkel med radien 8:

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

Python:

Funktionsdefinition för cirkelarea:

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

Parameter (formell Parameter)

Exempel: en cirkel med radien 8:

```
ar = a(8) #funktionen a anropas
# variablen ar tilldelas värdet som returneras
print(ar)
print(a(5.3))
```

Argument (aktuell Parameter)

Funktioner: programexekeringen (se F3_ex2.py)

1.

```
# 1. import-sater
import math

# 2. funktions-definitoner
def a(radie):
    ar = math.pi*radie*radie
    return ar

def rätblocksvolym(a, b, c): # OBS! En funktion kan ha flera parametrar!
    return a*b*c

# 3. Funktionsanrop, variabeldeklarationer, etc
area = a(5)    # variabeln area heter inte likadant som den returnerande variabeln a
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 (se F3_ex2.py)

```
import math

def a(radie):    Variablen radie sätts till 5
    ar = math.pi*radie*radie
    return ar

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

2.

```
#Funktionsanrop:
area = a(5)    # variabeln area heter inte likadant som den returnerande variabeln ar
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 (se F3_ex2.py)

```
import math


def a(radie):
    ar = math.pi*radie*radie
    return ar


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

3.

← Vad händer efter satsen return ar?

1. Programmet kör funktionen rätblocksarea
2. Programmet hoppar tillbaka till area=a(5)
3. Programmet hoppa till satsen under: print(’Cirkelns area är’, area)

```
#Funktionsanrop:
area = a(5)    # variabeln area heter inte likadant som den returnerande variabeln ar
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))
```

Programexekeringen (se F3_ex2.py)

```
import math

def a(radie):
    ar = math.pi*radie*radie
    return ar

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

3.

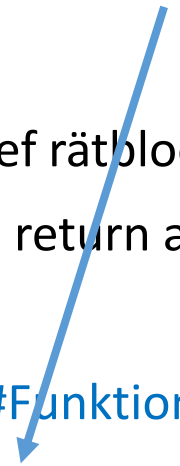
← Vad händer efter satsen return ar?

1. Programmet kör funktionen rätblocksarea
2. **Programmet hoppar tillbaka till area=a(5)**
3. Programmet hoppa till satsen under:
   print(’Cirkelns area är’, area)

```
#Funktionsanrop:
area = a(5)    # variabeln area heter inte likadant som den returnerande variabeln ar
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))
```

Programexekeringen (se F3_ex2.py)

```
1  import math
2
3  def a(radie):
4      ar = math.pi*radie*radie
5      return ar
6
7  def rätblockarea(a, b, c):
8      return a*b*c
9
10 #Funktionsanrop:
11 area = a(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))
14 print('Ett rätblock med måtten 7.1, 3.5 och 8.9 har arean', rätblockarea(7.1, 3.5, 9))
```

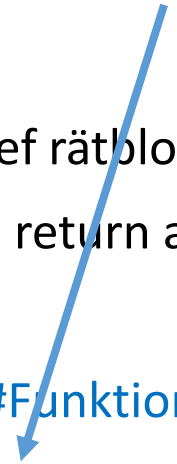


4. **VÄRDET returneras (inte bokstäverna ar), ges till variablen area**

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

Programexekeringen (se F3_ex2.py)

```
1  import math
2
3  def a(radie):
4      ar = math.pi*radie*radie
5      return ar
6
7  def rätblockarea(a, b, c):
8      return a*b*c
9
10 #Funktionsanrop:
11 area = a(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))
14 print('Ett rätblock med måtten 7.1, 3.5 och 8.9 har arean', rätblockarea(7.1, 3.5, 9))
```



4. **VÄRDET returneras (inte bokstäverna ar), ges till variablen area**

1. Vilken/vilka av följande rader kan area=a(5) INTE flyttas till:
rad 2, 6, 9 eller 15?
2. Varför inte? Funktionsdef ska läsas först
3. Vilken/vilka rad(er) skulle fungera men är opassande? 6 och 9: Ful kod!

Ha koll på programmet (debugga)

Mitt program krånglar.

a) Ger felmeddelande:

1. Läs felmeddelandet (kan vara svårt). Felmeddelanden kan peka på raden (raderna) med felet (felen).
2. Kommentera bort kod om du är osäker på var felet faktiskt är.

   Tonny: Edit -> Comment out (Alt+3 sen Alt+4)
3. Läs din kod i exekverings-ordning. Skriv ev på papper variablers värde efterhand.

b) eller ger fel svar:

1. Lägg in print-satser: `print('I funktionen calc. x=',x)`
2. Kommentera bort kod

Använd Debuggern!

Funktioner utan return-sats (returnerar *None*) Se F3_ex3

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

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

Ha koll på programmet (debugga)

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

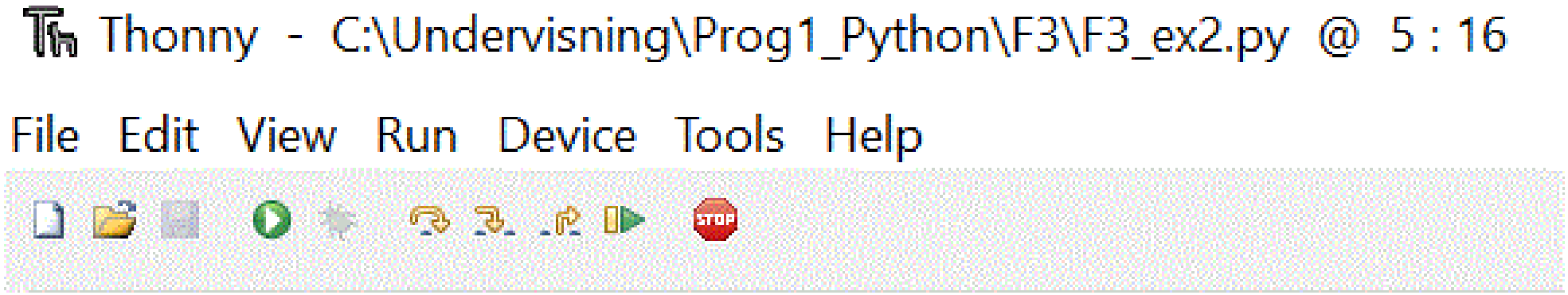
Sen **F6** (snabb) eller **F7** (detaljer) eller (**F8** kör som vanligt)

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



(Överkurs, men…) Hantera inmatningsfel (exceptions) Se F3_4d

se även F3_4b, F3_4c, F3_4d, (Lektion 4)

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

n = input('Vem har namnsdag idag? ’)

# Be användaren mata in ett tal till ett giltigt heltal matats in:
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 0, 1, … parametrar. Defaultparametrar. Se F3_ex4

```
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:
lov()
n = input(’Vem har namnsdag idag?’)
ant = int(input(’Hur många gånger vill du hurra? ’))
namnsdag(ant,n); print()
namnsdag(ant) #Vad händer här?
```

Funktioner kan ha 0, 1, … parametrar. Defaultparametrar. Se F3_ex4

```
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:
lov()
n = input(’Vem har namnsdag idag?’)
ant = int(input(’Hur många gånger vill du hurra? ’))
namnsdag(ant,n); print()
namnsdag(ant) #Vad händer här?
```

FRÅGA: Hur många gånger anrops funktioner när programmet körs?

2

6

10

13

15

Funktioner kan ha 0, 1, … parametrar. Defaultparametrar. Se F3_ex4

```
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:
lov()
n = input(’Vem har namnsdag idag?’)
ant = int(input(’Hur många gånger vill du hurra? ’))
namnsdag(ant,n); print()
namnsdag(ant) #Vad händer här?
```

FRÅGA: Hur många gånger anrops funktioner när programmet körs?

2

6

10

13

15

Funktioner kan ha flera defaultparametrar. Se F3_ex5.py

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’)
```

(Överkurs…) Okänt antal parametar: *args. Se Ex F3_ex4e.py

```
def my_friends(*names):
    for n in names:
        print(n, 'is my friend!’)

my_friends(‘Anna’)
# Utskrift:
# Anna is my friend

my_friends('Ada’, 'Bo', 'Cecil’)
# Utskrift:
# Ada is my friend!
# Bo is my friend!
# Cecil is my friend!
```

Funktioner: Scope och lokala variabler F3_ex6b

```
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, från här
9  m = month_lived()
10 print(’You have lived about ', m , ' months’)
11
12 print(And here month=', month)
# Fråga: vad blir utskriften från rad 7, 10 och 12?
```

Scope F3_ex6

```
def first_func(x): # Parametern x är en lokal variabel
    y=1 # y är en lokal variabel som finns bara i funktionen, pekar initialt på 2
    x = y+3 # Den lokala variablen x tilldelas en ny minnesadress med 6
    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(’Now x =',x)
# print(y) FEL. y bara definierad i sitt scope, dvs i funktionen first_func
print('second_func: ', second_func())
print(’But now x =’,x)
```

Funktioner som parametrar F3_ex6c.py (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!'))
```

Snygg, lättläst kod som följer konventionen.

```
import …

def hypotenuse_rectangle(k1, k2):
    """Takes in length of… returns …"""
    ….

def interest_per_month(s, i, m):
    """Takes in … , returns …"""
    ….


#code here:
variables
function calls…
~~X1=… # bad variable name~~
~~X2=…~~
~~X3=…~~
```

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

Två tomma rader mellan funktions- definitionerna. Doc-strings """… """

Engelska, helst.

Liten bokstav i namn, _ för tydlighet. clear_names.

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. Ex: 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 metodanrop** 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) F3_ex7

```
#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) F3_ex7a

```
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)
```

Funktionen swap (byt värden) F3_ex7b

```
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


x = 7
y = 5
tup = swap(x,y)  # x ska bli 5 och y 7
x = ….  ;  y = …
print(’Swaped: x=',x, ' ' 'y=',y)
```

Funktionen swap (byt värden) F3_ex7c

```
def swap(a,b):
    temp = a   # spara a i temp
    a = b      # kopiera b till a
    b = temp   # kopiera temp till b
    return [a,b] # returnerar en referens till en lista


x = 7
y = 5
lis = swap(x,y) # lis pekar på en lista
x=lis[0] #tup[0]
y=lis[1] #tup[1]
print('Swaped: x=',x, ' ' 'y=',y)
```

Tack för idag!