UPPSALA UNIVERSITET

Parametrar, returvärden, tupler och felsökning

Carl Nettelblad

2020-01-28

Info

- Tänk på deadlines OU1 och OU2
- OU4 har fått sitt nya innehåll
  - Kan ske småjusteringar
  - Deadline 2020-02-27/2020-02-28
    - Skottår, inte sista februari

- Under början av andra timmen den 6 februari håller doktoranden Kristiina Ausmees en kort gästföreläsning om hur hon använder AI-modeller i Python för att analysera genetisk variation

UPPSALA UNIVERSITET

Uttrycks typ

- Varje *uttryck* i Python har alltså en typ
  - Beror på typerna på ingående uttryck
  - En variabels typ beror på variabelns värde
  - En variabel kan byta typ vid tilldelning

```
a = 3
a = 'Hej'
```

  - Python är dynamiskt typat (en variabels typ kan ändras under körning, alltså dynamiskt)

UPPSALA UNIVERSITET

Värden och referenser

- Python har två sorters typer
  - Primitiva typer
    - ”Värden”
    - Heltal, flyttal, sanningsvärden och strängar
  - Alla andra typer
    - Till exempel listor

- En variabel innehåller antingen ett värde
  - Inklusive det speciella värdet None
  - Eller en *referens* till ett *objekt* av en annan typ

UPPSALA UNIVERSITET

Växla värden och referenser

- ”Python Tutor” för visualisering

```
a = 3
b = a
a = [1,2,3]
b = a
b[2] = 5
a = a + b
```

UPPSALA UNIVERSITET

for-slinga

- Vi har sett `while`
- Loopa över en lista kunde skrivas

```
i = 0
while i < len(a):
      print(a[i])
      i += 1
```

- Finns ett bättre alternativ!

```
for x in a:
      print(x)
```

UPPSALA UNIVERSITET

`break`, `pass`, `continue`

- `break` avslutar en slinga (i förtid)
  - Återstående värden passeras inte, `while`-villkoret kontrolleras inte igen för en `while`-slinga
- `continue` hoppar till *nästa* steg i en slinga
  - Resten av kodet i blocket körs inte för den nuvarande *iterationen*
  - Hämtar nästa värde och början om från början för `for`, testar villkoret igen för `while`
- `pass` gör **ingenting**, används i tomma block

UPPSALA UNIVERSITET

Lista med heltal

```
listan = []
for el in data.split(','):
    listan.append(int(el))
```

UPPSALA UNIVERSITET

`map` till vår räddning

- `map(func, x)`
  - Ta in ett itererbart objekt x (som en lista)
  - Returnera ett nytt itererbart objekt
  - I det nya objektet är varje värde returvärdet från funktionen `func` på motsvarande element i `x`
- Så om `x` ger elementen `'39'`, `'12'`, `'68'` ger `map(func, x)` elementen `func('39')`, `func('12')`, `func('68')`

UPPSALA UNIVERSITET

map

```
result = map(int, data.split(’,’))
listan = list(result)

for el in map(int, data.split(’,’)):
      print(el)
```

Vad är typen för result?

UPPSALA UNIVERSITET

List comprehensions

```
listan = [int(x) for x in data.split(’,’)]
```

- Snabbt sätt att skapa en lista genom att beskriva hur varje element i listan ska skapas

UPPSALA UNIVERSITET

Fler operationer på listor

- Samma element upprepade flera gånger

```
a = [1,2] * 3
b = a
a = a + [3,4] + a
```

- Vad är `b` på slutet?
- Vilka objekt skapas? Vilka objekt refererar variablerna till?

UPPSALA UNIVERSITET

Radera element i lista

```
c = b
del b[1]
```

UPPSALA UNIVERSITET

En egen funktion

```
def longestString(str1, str2):
        if len(str1) > len(str2):
                return str1
        else:
                return str2
```

- Inled med `def`, funktionsnamn och noll, en eller flera parametrar
  - Glöm inte kolon och indrag
- Resultatet av funktionen skrivs med `return`
  - Värdet skickas tillbaka/returneras och funktionen avslutas

UPPSALA UNIVERSITET

En funktion utan returvärde

```
def printLongestString(str1, str2):
    if len(str1) > len(str2):
        print(str1)
        return
    print(str2)
```

- Man kan avsluta en funktion utan att returnera något
  - Genom att använda `return` utan värde
  - Genom att komma till sista raden, utan `return`

UPPSALA UNIVERSITET

Docstrings

- I *början* av varje egen funktion vill vi förklara hur den ska användas.
  - Är det något man måste veta om vad den gör? Vad parametrarna betyder? När den inte bör användas?
  - Använd tre citationstecken.

```
def longestString(str1, str2):
        """Returnerar den längsta strängen av två.
           Om lika långa returneras str2."""
        if len(str1) > len(str2):
                return str1
        else:
                return str2
```

Funktioner och parametrar

```
def square(x):
      x = x * x
      return x

x = 9
y = square(x)
```

- Vad har x och y för värden när koden tar slut?

  www.menti.com, kod 69 43 68

UPPSALA UNIVERSITET

Slutsats

- En parameter är en variabel *inne i funktionen*
  - En *lokal* variabel
- Vad den heter och vad den har för värde inne i funktionen spelar ingen roll utanför funktionen
  - Parametern kan råka ha samma namn utanför, men behöver inte ha det

- Om en variabel inte finns lokalt kan funktionen *läsa* variabler utanför:

```
def square():
        return x * x

x = 9
print(square())
```

UPPSALA UNIVERSITET

Standardvärden

```
def squareandmore(x, exponent=2):
    x = x ** exponent
    return x


a = 9
b = squareandmore(a)
c = squareandmore(a,3)
```

- Om man bara anger en parameter används exponenten 2.
- Gör det lätt att lägga till flexibilitet i en funktion utan att ändra gammal kod.

UPPSALA UNIVERSITET

Namngivna parametrar i anrop

```
def squareaddandmore(x, term=0, exponent=2):
      x = x ** exponent + term
      return x

squareaddandmore(5, exponent=3)
squareaddandmore(5, exponent=3, term=1)
squareaddandmore(5, 1, 3)
squareaddandmore(term = 1, exponent=3)
```

UPPSALA UNIVERSITET

Namngivna parametrar i anrop

- Om en parameter har standardvärde kan den hoppas över genom att man anger namnet på nästa
- Man kan växla ordningen på namngivna parametrar
- Man kan aldrig utelämna en parameter som saknar standardvärde

UPPSALA UNIVERSITET

Moduler

`import turtle`

- Då kan vi sedan använda delar från turtle genom att skriva `turtle.`*`namn`*, till exempel `turtle.Turtle()` för att skapa ett `Turtle`-objekt
- Turtle är alltså en ny typ som definieras i modulen turtle

`from turtle import *`

- Då kan vi skriva `Turtle()` direkt (inget `turtle.` först)

`from turtle import Turtle`

- Vi kan ange exakt vilka namn vi vill använda, bra om modulen är stor, eller om flera moduler man använder har delar med samma namn

`import turtle as t`

- Så kan man skriva `t.Turtle()`, `t` blir ett alias för `turtle`
- En del modulnamn är väldigt långa, koden kan bli tjatig

UPPSALA UNIVERSITET

Returnera flera värden

```
def squareandcube(x):
      return x**2, x**3

a = squareandcube(9)
```

UPPSALA UNIVERSITET

Tupler

- Förutom listor har Python ett annat viktigt sätt att hantera kombinationer av flera värden
  - Tupler
- I en lista kan elementen ändras
- En listas storlek kan ändras

- En tupel är oföränderlig
  - Men om en tupel innehåller en referens kan förstås objektet som den refererar till förändras

UPPSALA UNIVERSITET

Skapa en tupel

- När vi skriver `return x**2, x**3` betyder det ”skapa en ny tupel med två element med värdena `x**2` och `x**3` och returnera den tupeln”
- Ofta skapar vi en tupel genom att bara skriva flera värden, med komma
- I ett funktionsanrop måste vi sätta tupeln inom parentes

```
print((16,64),’hej’)
print(16,64,’hej’)
```

- I ena fallet har vi två parametrar, där den första är en tupel av två heltal

Ta ut element ur en tupel

- a[0] tar det första elementet ur en sekvens, oavsett om det är en sträng (första tecknet), en lista eller en tupel

- En sekvens kan också ”packas upp”
- Skriv en tilldelning där sekvensen (strängen/listan/tupeln) är högerledet och vänsterledet är lika många nya variabler

```
square, cube = squareandcube(9)
hchar, echar = ’he’
```

- Väldigt vanligt sätt att hantera funktioner med flera returvärden
  - Användes i exemplen med `zip` och `enumerate`

```
hchar, echar = ’hej’
```
(fungerar ej, fel antal)

UPPSALA UNIVERSITET

Nästlade strukturer

- En lista är ett objekt vilket som helst.
- En tupel är ett objekt.
- En lista kan innehålla referenser till andra objekt.
- En tupel kan innehålla referenser till andra objekt.
- Vi kan alltså ha en lista

```
a = [('Sverige', 'Stockholm'), ('Frankrike', 'Paris')]
```

- Eller en tupel med listor:

```
b = (['Sverige', 'Frankrike'], ['Stockholm', 'Paris'])
```

- Eller en tupel med tupler

```
c = ('+', ('*', 3, 4), ('/', 1, 7))
```

UPPSALA UNIVERSITET

Nästlade strukturer

- Eller en lista med sig själv

```
a = []
a.append(a)
```

- Eller en tupel med samma lista två gånger…

```
a = [42]
b = (a, a)
a.append(43)
```

Nästlade strukturer

- Kan vara bra att titta på visualiseringer i Python Tutor eller rita själv.
  - Python Tutor finns på webben `pythontutor.com`
  - Kan integreras i Jupyter
    - Installera paketet `metakernel`
    - I varje Notebook (finns andra sätt)

```
from metakernel import register_ipython_magics
register_ipython_magics()
```

    - Sedan `%%tutor` i beräkningscellen

UPPSALA UNIVERSITET

Lexikon

- Vi vill ofta lagra värden som hör ihop
  - Till exempel betyg för varje namn

- Vi kunde göra varje namn till en variabel

```
amina = 5
johanna = 4
gregor = 3
carl = None
```

- Inte så flexibelt, ska vi skriva om programmet om en ny student registrerar sig på kursen?

UPPSALA UNIVERSITET

Studentresultat

- Vi kan ha en lista med namn och en lista med betyg.

```
namn = [’Amina’, ’Johanna’, ’Gregor’, ’Carl’]
betyg = [5, 4, 3, None]
```

- Hur gör vi för att hitta Gregors betyg?
  - Iterera över hela listan `namn` med en slinga
  - Slå upp samma index i `betyg`

- Riskabelt att hålla listorna i synk
  - Vad händer om man lägger till/tar bort namn?

UPPSALA UNIVERSITET

Slå upp student, lite bättre

```
for namnet, betyget in zip(namn, betyg):
      if namnet == ’Gregor’:
            break
```

- Vi måste fortfarande gå igenom alla element
- Men går igenom dem som tupler

- Borde vi *lagra* dem som tupler?

UPPSALA UNIVERSITET

Betygslista som tupler

```
betyg = [('Amina', 5), ('Johanna', 4),
         ('Gregor', 3), ('Carl', None)]

for namnet, betyget in betyg:
      if namnet == 'Gregor':
            break
```

Lexikon

- Speciell typ för att lagra tvåtupler (par) av *nycklar* och *värden*
  - `dict` i Python (från *dictionary*)
- En nyckel kan förekomma högst en gång i ett lexikon
  - Så vi får hoppas att inga studenter har samma förnamn i vårt exempel

```
betyg = [('Amina', 5), ('Johanna', 4),
         ('Gregor', 3), ('Carl', None)]
betygDict = dict(betyg)
```

eller skapa den direkt

```
betygDict = {'Amina' : 5, 'Johanna' : 4,
             'Gregor' : 3, 'Carl' : None}
```

Lexikon fungerar som listor

- I stället för index använder vi nycklarna

```
betyget = betygDict['Gregor']
```

- Nyckeln måste finnas (ger fel)

```
betyget = betygDict['Max']
```

- Nyckeln måste vara exakt lika, följande värden finns *inte* i lexikonet

```
betygDict['gregor']
betygDict['Gregor ']
```

- Vi kan ändra eller lägga till med indexering

```
betygDict['Carl'] = 3
betygDict['Elsa'] = 4
```

- Vi kan ta bort en nyckel (och dess värde)

```
del betygDict['Carl']
```

UPPSALA UNIVERSITET

for-slingor med lexikon

- Bara nycklar (se upp med ordningen!)

```
for el in betygDict:
    print(el)
```

- Bara värden

```
for el in betygDict.values():
    print(el)
```

- Tupler med nycklar och värden

```
for el in betygDict.items():
    print(el)
for namn, betyg in betygDict.items():
    print(namn, betyg)
```

UPPSALA UNIVERSITET

Vad kan vara en nyckel?

- Många typer kan vara nycklar
  - Strängar
  - Heltal
  - Flyttal
  - Sanningsvärden
  - Tupler (beroende på vad de innehåller)
  - Turtlar!
  - *Inte listor*
- Typen måste stödja vissa typer av jämförelser
  - Att använda en föränderlig typ som nyckel kan ge problem
- Strängar och heltal ojämförligt vanligast

UPPSALA UNIVERSITET

Allt är ett lexikon

- Variablerna är också ett lexikon!
- Du kan hitta alla lokala variabler genom att anropa funktionen locals()
- Och ändra på dem…

```
x = 3
print(locals()['x'])
locals()['x'] = 99
print(x)
```

UPPSALA UNIVERSITET

Allt är ett lexikon

- Alla objekt har också ett lexikon med sin information
  - Det kan vara intressant att titta på `t.__dict__` för en padda

- En modul är också ett lexikon

```
import math
print(math.__dict__)
```

- Man ska normalt INTE modifiera eller titta på dessa
  - Men flexibelt i vissa fall

UPPSALA UNIVERSITET

Hur ska jag tänka?!

- Att lösa ett problem är svårt.
- Försök att dela upp det i delproblem.
  - Kan jag säga ”först behöver jag göra A”, ”sedan behöver jag göra B”?
  - Kan jag till och med skissa koden:

    ```
    A()
    B()
    ```

  - Nästa steg blir att komma fram till parametrar och returvärden

UPPSALA UNIVERSITET

I en enda funktion

- Vilka fall finns det?
  - Ska alltid samma saker hända?
  - Finns det varianter?
  - Vilka parametervärden är rimliga?
    - Kan jag komma på några värden där jag vet vad svaret ska bli?
    - Finns det några ”extrema” värden som är bra att testa
      - Negativa tal, 0, tomma listor, tomma strängar, …
      - Oerhört vanligt med buggar på grund av sådant!

UPPSALA UNIVERSITET

`isprime`

- Vi vill skriva en funktion `isprime` som tar reda på om ett tal är ett primtal.
  - Steg 0: Vad är ett primtal?
    - Ett icke-negativt heltal större än 1 som inte har några andra heltalsdelare än 1 och sig själv.
  - Steg 1: Den enda parameter som behövs är talet självt, x. Vi förväntar oss att det är ett icke-negativt heltal. Vi tänker inte kontrollera det. (FARLIGT, men praktiskt!)
  - Steg 2: Vi berättar svaret genom att returnera en bool med värdet `True` om det är ett primtal och `False` annars

UPPSALA UNIVERSITET

Börja skissa lösningen

```
def isprime(x):
      """Tar reda på om x är ett primtal."""
      if x <= 1:
            return False
      # Kolla om det finns någon delare



      # I sista hand är det ett primtal
      return True
```

UPPSALA UNIVERSITET

En bit på vägen

- Den här funktionen returnerar redan rätt svar för alla primtal!
- Den svarar också rätt för 1 och andra lägre tal.
- Tyvärr fel svar för alla andra tal.

- Nu måste vi ”bara” testa alla andra tänkbara delare.
  - När vi har en mängd element vi vill göra samma sak för använder vi ofta en
    - `for`-slinga
    - Alla tal från 2 upp till talet närmast före x

UPPSALA UNIVERSITET

for-slingan tar form

```
def isprime(x):
    """Tar reda på om x är ett primtal."""
    if x <= 1:
        return False
    for i in range(2, x):
        pass
        # TODO: Kolla om i är en delare
    return True
```

UPPSALA UNIVERSITET

Slingans innehåll

- Den nya funktionen ger samma svar som den gamla.
- Men vi har begränast problemet vi ska lösa. Från att ta reda på om `x` har någon delare behöver vi bara testa om just talet `i` är den delare till `x`.
- Går att skriva som

```
if x//i*i == x:
```

- Men bättre som

```
if x%i == 0:
```

- Så fort någon delare har hittats kan vi returnera `False`

UPPSALA UNIVERSITET

for-slingan tar form

```
def isprime(x):
      """Tar reda på om x är ett primtal."""
      if x <= 1:
            return False
      for i in range(2, x):
            if x%i == 0:
                  return False
      return True
```

UPPSALA UNIVERSITET

Korrekt, men…

- Hur funkar det att köra `isprime(1000000007)` ?
- `%time` i ipython/Jupyter
- Om `i` är en delare till `x` är även kvoten `j = x // i` en delare till `x`.
  - Vi vet att `i * j == x`
  - Antag att `i > sqrt(x)`
  - I så fall måste `j < sqrt(x)`
- Alltså räcker det med att testa delare upp till kvadratroten av `x`.
  - Sparar tid!

UPPSALA UNIVERSITET

Första försöket

```
def isprimefast(x):
      """Tar reda på om x är ett primtal."""
      if x <= 1:
            return False
      for i in range(2, int(math.sqrt(x))):
            if x%i == 0:
                  return False
      return True
```

Regressionstester

- Vi hade `isprime` som vi litade på.
- `isprimefast` ska göra samma sak.
- Svårt att skriva kod som garanterar att de alltid gör samma sak, men lätt att testa för några/många fall:

```
for i in range(100):
    if isprime(i) != isprimefast(i):
        print(f'Skillnad för {i}')
        break
```

- Även om man ändrat någon del kan det vara klokt att försöka kolla att allt annat är likadant som förut.

UPPSALA UNIVERSITET

Ojdå?

- Vi får försöka ta reda på vad som händer.
- Bilda hypoteser.
  - Eftersom vi får svaret `True` när det borde ha varit `False` verkar det som någon delare inte testas ordentligt.
  - Kontrollera vilka delare som testas!
    - Läsa koden
    - Köra koden med fler kontroller

UPPSALA UNIVERSITET

Med felsökning

```
def isprimefast(x):
      """Tar reda på om x är ett primtal."""
      if x <= 1:
            return False
      for i in range(2, int(math.sqrt(x))):
            print(f'testar {i}')
            if x%i == 0:
                  return False
      return True
```

UPPSALA UNIVERSITET

Vad ser vi?

- Den stannar för tidigt.
- Alltså får vi titta på vår `for`-slinga

```
for i in range(2, int(math.sqrt(x))):
```

- Vad är `int(math.sqrt(15))`?
- Intervallet för `range` är högerexklusivt
- Avrunda uppåt!

UPPSALA UNIVERSITET

Avrunda uppåt

```
def isprimefast(x):
      """Tar reda på om x är ett primtal."""
      if x <= 1:
            return False
      for i in range(2, math.ceil(math.sqrt(x))):
            if x%i == 0:
                  return False
      return True
```

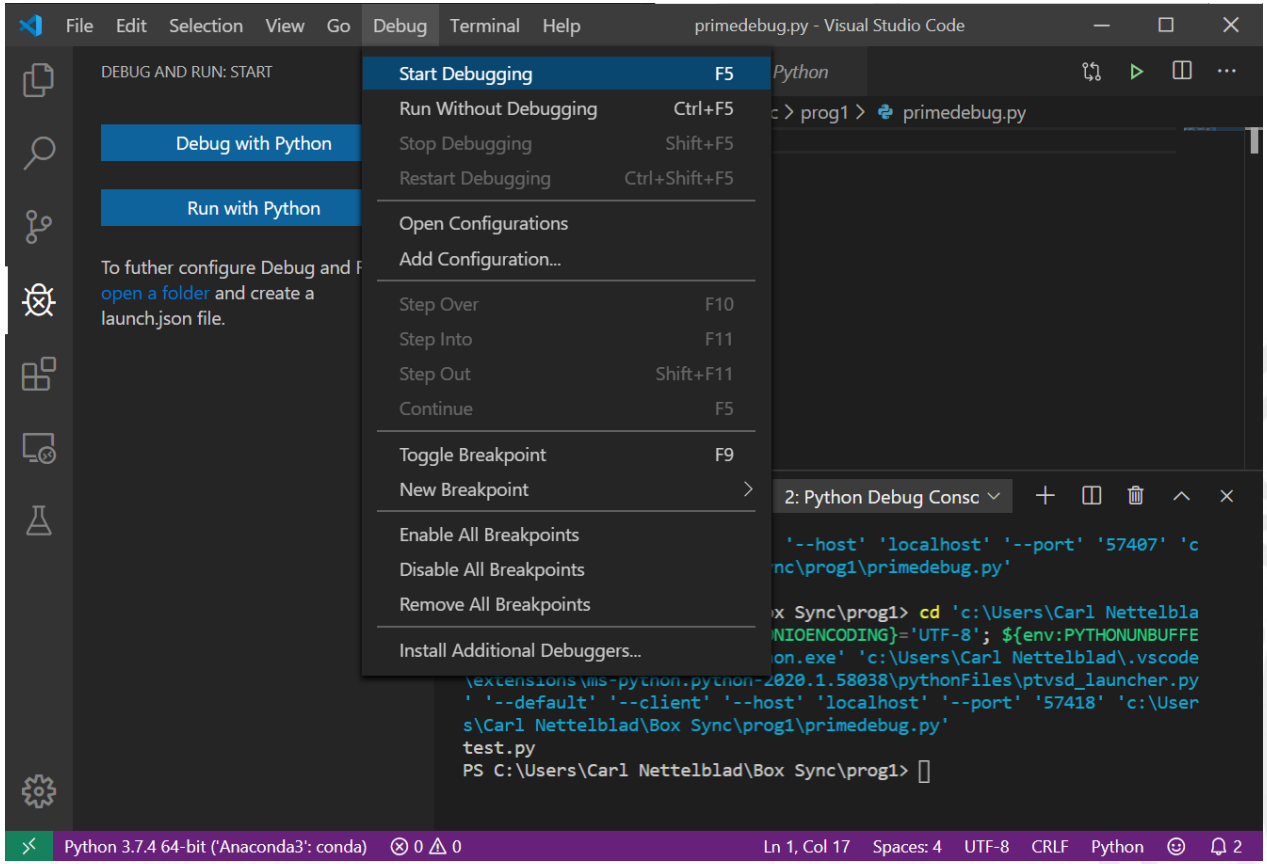
UPPSALA UNIVERSITET

Regressionstesta igen

- Fortfarande problem?!
- Kvadrater av primtal…
- Låt oss använda en *debugger*

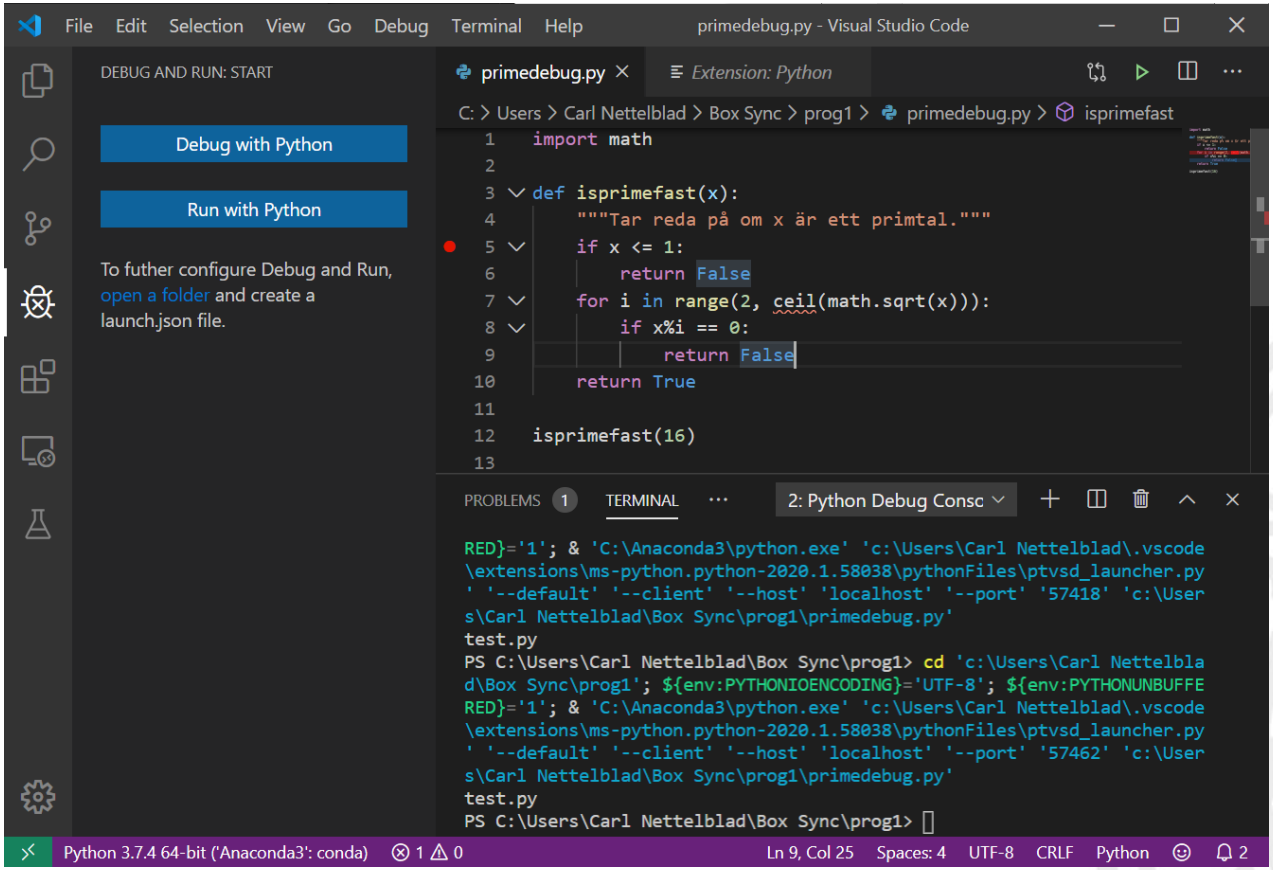
UPPSALA UNIVERSITET

Debuggern i VS Code



UPPSALA UNIVERSITET

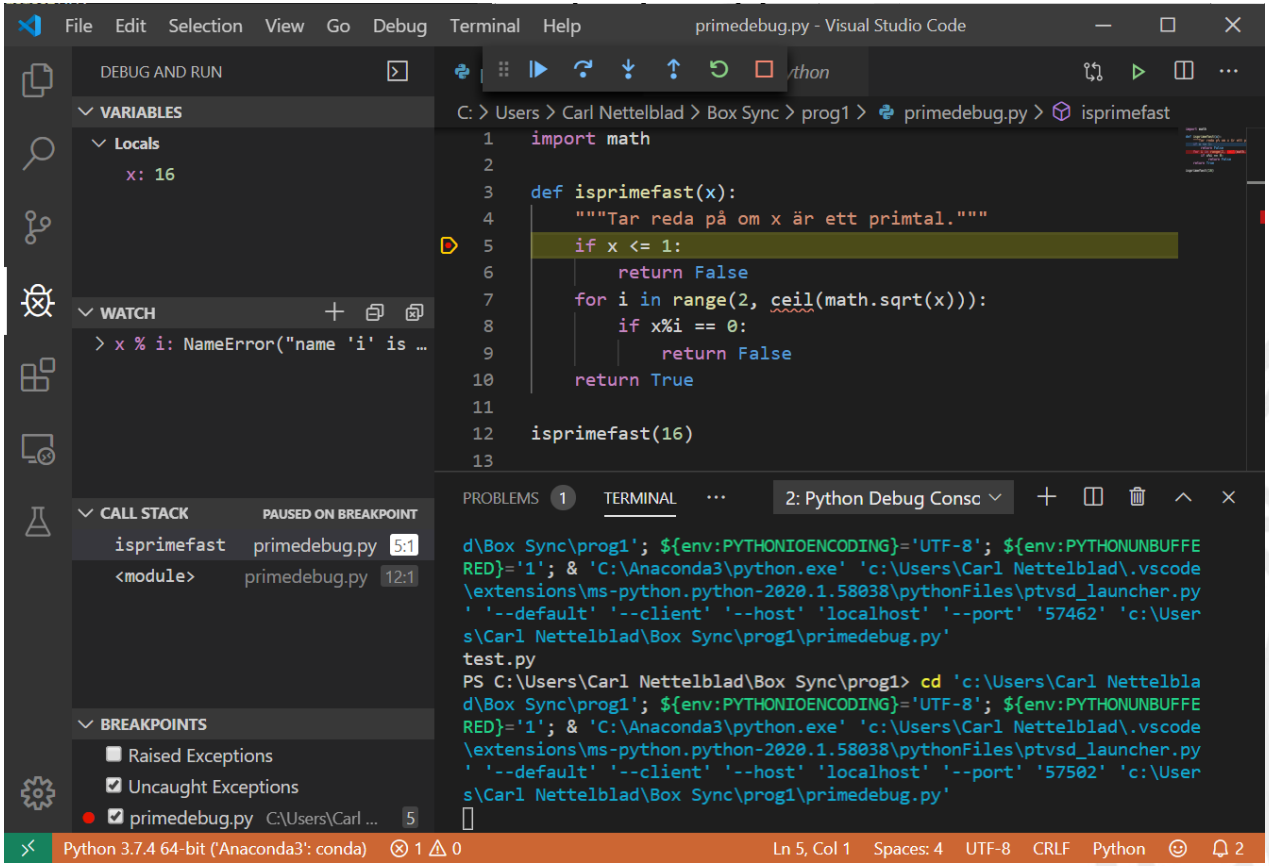
Debuggern i VS Code



- Klicka för att skapa *brytpunkt* (röda pricken).
- Starta felsökning igen.

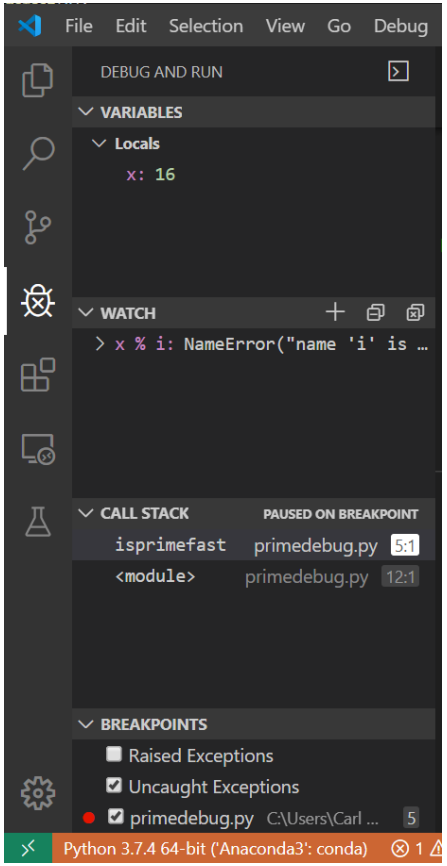
UPPSALA UNIVERSITET

Debuggern i VS Code



- Programmet stannar när brytpunkten nås.

Vänsterfältet



- Till vänster ser vi värden på alla variabler.
- Man kan ändra variabeln om man vill.
- ”Watches”, uttryck som vi ser aktuellt värde för. Här har vi lagt till `x % i`. Det ger ett fel, eftersom variabeln i inte finns än.
- Anropsstacken, vilken funktion kör vi i och var anropades den ifrån.

Körkontroll

- Från vänster till höger:
  - Kör vidare (F5, till nästa brytpunkt)
  - Step Over (F10, kör nästa kodrad, gå inte in i anropade funktioner)
  - Step Into (F11, kör nästa kodrad, stanna inne i anropade funktioner)
  - Step Out (Skift+F11, kör nästa kodrad i den anropande funktionen)

UPPSALA UNIVERSITET

Vad hände?

- Kvadratroten av 25 är exakt 5

```
math.ceil(math.sqrt(x)) == 5
```

- Det var inte vad vi tänkte. Gränsen för range bör vara *största-talet-som-behöver-testas* + 1
- `int(math.sqrt(x)) + 1`

UPPSALA UNIVERSITET

Regressionstesta igen

- Yes!

UPPSALA UNIVERSITET

Kvadrera elementen i en lista

```
def square(x):
      x = x * x
      return x


a = [3,5]
b = square(a)
```

- Fungerar inte, felmeddelande

UPPSALA UNIVERSITET

Kvadrera elementen i en lista

```
def square(x):
      for i in x:
            i = i * i
      return x


a = [3,5]
b = square(a)
```

- Fungerar inte, `i` är en *kopia* på värdet
- Ingen kvadrat sparas

UPPSALA UNIVERSITET

Kvadrera elementen i en lista

```
def square(x):
      for j,i in enumerate(x):
            x[j] = i * i
      return x

a = [3,5]
b = square(a)
```