Tio lektioner i matteprogrammering 1

Med grafritning och numeriska metoder

Ett övningshäfte till läroboken:

Matematik åk 7-9 och Matematik 1 (a, b, c)

Titel: Tio lektioner i matteprogrammering 1
Ett övningshäfte till läroboken *Koda matte med Python*
www.kodamatte.se
boken@kodamatte.se

ISBN: 978-9-197-42044-0

Författare: Taifun Alishenas
info@taifun.se

Tel: 08 - 792 36 28
www.techpages.se

Tryckeri: Eprint, Stockholm
Januari 2020

Innehåll

Varför ett övningshäfte?

I Skolverkets nya läroplaner ingår programmering i skolans matematikundervisning. På vilket sätt detta ska realiseras är dock en gåta för de flesta matematiklärare som varken är utbildade i programmering eller har någon erfarenhet av att koppla det nya ämnet till den redan befintliga, tunga mattekursen.

Läroboken *Koda matte med Python* som publicerades hösten 2018 var ett första försök att brottas med denna svårlösta ekvation: att ge en introduktion till programmering *och* kombinera den nya kunskapen med lämpliga delar av skolmatematiken (`kodamatte.se`). Responsen från läsarna lät inte vänta på sig: man var bekymrad över lärobokens användning i klassrummet både med avseende på volym och upplägg. Man föreslog ett tunnare material med *lektionsupplägg* istället för den traditionella kapitelstrukturen. Så föddes idén med ett övningshäfte bestående av lektioner som verkligen kan genomföras under *ett* lektionspass.

Varje lektion består av ett löst exempel som direkt kan matas in och testas i Python. Viss matematisk förberedelse och korta kodförklaringar ska underlätta förståelsen och ersätta långdragen teori. Ett antal frågor samt övningar kring samma tema följer. På så sätt blir lektionen en *laboration* i programmering för för skolans mattekurser. Lösningsdelen innehåller svar på frågorna samt fullständiga lösningar till alla övningar.

Det nya lektionsupplägget visade sig ha vissa konsekvenser:

- Vad gäller programmeringen kan inte alla nya koncept förklaras fullt ut. Intuitiv förståelse prioriteras framför fullständig teoretisk klarhet.
- Även om lektionsupplägget har en logisk ordning, följer övningshäftet inte upplägget i de mest använda kursböckerna i matematik. Istället görs ett urval med hänsyn till lämpligheten för programmering.
- Även om det är tänkt att varje lektion kan genomföras under *ett* lektionspass kan tidsåtgången variera pga praktiska omständigheter. Ingen regel utan undantag: den sista lektionen – **Lektion 10** som tar upp digital ekvationslösning med en numerisk iterationsmetod – behöver antagligen längre än ett lektionspass. I sådana fall kan man *börja* lösa uppgifterna under lektionstid och fortsätta ge dem som läxor.

Appendixdelen beskriver hur man laddar ned och installerar alla verktyg som behövs för att testa häftets exempel och lösa övningarna.

Ett komplement till detta övningshäfte är appen *Mattekollen* (`app.mattekollen.se`) som digitalt hjälpmedel både till programmeringen och som stöd för skolans mattekurser. Appen ger även tillgång till en mobil pythonmiljö där eleverna kan testa sina första försök att koda Python för att lösa matematiska problem.

Tio lektioner i matteprogrammering kan delas ut som övningshäfte till eleverna och användas för självständigt arbete i klassrummet både i högstadiet (åk 7-9) och gymnasiets Matematik 1.

Övningshäftet är helt oberoende av läroboken *Koda matte med Python*.

Kapitel 1

Taluppfattning

Lektion 1

Prioritetsordning (PEMDAS)

Tennisbollen

En tennisboll kastas rakt upp med en hastighet på $v = 14$ meter/sekund.

Beräkna bollens höjd h efter $t = 2$ sekunder med formeln:

$$h = v \cdot t - (1/2) \cdot 9{,}8 \cdot t^2$$

Manuell lösning

Sätt in $v = 14$ och $t = 2$ i formeln ovan och räkna enligt prioriteringsreglerna **PEMDAS**:

$$h = 14 \cdot 2 - (1/2) \cdot 9{,}8 \cdot 2^2$$

1) Parantesen först: $14 \cdot 2 - 0{,}5 \cdot 9{,}8 \cdot 2^2$

2) Potensen sedan: $14 \cdot 2 - 0{,}5 \cdot 9{,}8 \cdot 4$

3) Multiplikation: $28 - 0{,}5 \cdot 9{,}8 \cdot 4$

$28 - 19{,}6$

4) Subtraktion sist: $8{,}4$

Tennisbollen når efter 2 sekunder höjden 8,4 meter.

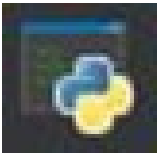
Python som smart kalkylator

Python_Calc

Öppna Python genom att klicka på pythonikonen →

Mata in följande och tryck på Enter:

```
>>> 14 * 2 - (1/2) * 9.8 * 2**2
8.399999999999999
>>>
```



22** är pythonkoden för 2^2.

Detta förutsätter att du antingen har tillgång till en valfri pythonmiljö på webben (t.ex. app.mattekollen.se) eller har installerat Python på din dator enligt appendix **A** (sid 101).

I utskriften ovan får vi förstås samma resultat som i den manuella lösningen: 8,4, fast avrundat. De 15 decimalsiffrorna i Pythons utskrift beror på att datorn jobbar med nollor och ettor. Den omvandlar alla tal till det binära talsystemet med basen 2. Så omräkningen från och till vårt decimala talsystem leder ofta till avrundningsfel som dock kan försummas.

Resultatet ovan visar att Python räknar enligt de gällande prioriteringsreglerna ***PEMDAS*** som är implementerade i Python, därför *"smart" kalkylator*. I övningarna finns fler exempel. Mer information om dessa regler finns i boken *Koda matte med Python* (sid 22).

Interactive mode

I koden **`Python_Calc`** på förra sidan matade vi in ett *aritmetiskt uttryck* och fick svaret direkt. Tekniken kallas för *interactive mode* och kan användas för all pythonkod, vilket gör det möjligt att använda Python bl.a. som en smart kalkylator. Interactive mode kan alltid avslutas med tangentsekvensen **`<ctrl>-Z`** följd av **`Enter`**, även om man inte får tillbaka prompten.

Frågor

1. Vad innebär ***PEMDAS*** ? Beskriv med egna ord.
2. Testa i Interactive mode `6 + 3 * 5` och förklara resultatet med ***PEMDAS***.
3. Vad ger `2**3` i Python? Vilken räkneoperation måste `**` vara koden för?
4. Vad är skillnaden mellan `(-2)**2`? och `-2**2`? Testa med Python och förklara.
5. Ett annat sätt att beräkna tennisbollens höjd (sid 9) är:

   ```
   14 * 2 - 9.8 * 2**2 / 2
   ```

   Vad blir resultatet i Python? Skriv om divisionen till bråkstreck för och räkna manuellt.

6. $12 / 2 \cdot (1 + 5)$ — Räkna parentesen först: $12 / 2 \cdot 6$, sedan: $12 / 12 = 1$?
   eller
   Räkna vänster till höger: $12 / 2 = 6$, sedan: $6 \cdot (1 + 5) = 6 \cdot 6 = 36$?

   a) Testa `12 / 2 * (1 + 5)` i Python. Svarar Python rätt? Följer Python ***PEMDAS***?
   b) Var är felet i det felaktiga svaret? Vilken regel i ***PEMDAS*** har inte beaktats?
   c) Förtydliga det ursprungliga uttrycket med parenteser för att förenkla lösningen.
   d) Hur skulle parenteserna sättas om man vill få det andra svaret?

Övningar

1101

Använd Python som smart kalkylator för att beräkna följande aritmetiska uttryck:

a) $7 + 4 \cdot 2$

b) $9 - 8 / 4$

c) $12 + 18 / 9 - 6$

d) $\dfrac{12 + 18}{9 - 6}$

1102

Räkna först utan Python och kontrollera sedan dina resultat med Python:

a) $5 + 3 \cdot 8 - 6$

b) $(5 + 3) \cdot (8 - 6)$

c) $3(6 - 4) + 2(5 - 2)$

d) $6(3 + 1 \cdot 2) - 4 \cdot 5$

1103

Ett taxibolag tar en fast avgift på 25 kr. Därefter kostar det 10 kr per km att åka. Beräkna med Python hur mycket det kostar att åka 20 km. Ställ upp ett aritmetiskt uttryck för taxan om man åker x km.

1104

Ett mobilabonnemang har en öppningsavgift på 1,50 kr. Sedan kostar det 25 öre per minut att ringa. Ställ upp ett aritmetiskt uttryck för kostnaden om man ringer x sekunder. Beräkna uttryckets värde för x = 59 i Python, dvs låt Python beräkna hur mycket det kostar att ringa i 59 sek.

1105

Beräkna följande aritmetiska uttryck utan digitalt verktyg. Kontrollera sedan dina resultat i Python.

a) $(-5)^2 - 3^2$

b) $\dfrac{2^2 + 3^2}{(2+3)^2}$

c) $\dfrac{(4-2)^2}{4^2 - 2^2}$

d) $(2^2)^3$

e) $2^{(2^3)}$

f) 2^{2^3} Förklara hur Python räknar.

1106

Låt Python beräkna och skriva ut följande uttryckets värde för $x = -1$:

$$4x^3 - 2x^2(2x + 6) + 7x(3 + 2x)$$

1107

a) Beräkna följande uttryck i Python:

$$(6^2 + 6^2 + 6^2) / 9$$

b) Kontrollera Pythons svar genom att förenkla uttrycket och beräkna det sedan.

Fullständiga svar och lösningar till frågorna och övningarna finns i lösningsdelen på sid 55.

I denna lektion använde vi *aritmetiska uttryck* bestående av siffror, operatorer och parenteser. I nästa lektion kommer vi att blanda in även bokstäver, dvs gå över till *variabler* och använda *algebraiska* uttryck. **PEMDAS** gäller även för algebraiska uttryck. Hittills har vi matat in all pythonkod i Interactive mode och kommer att göra det även ett tag till. Först f.o.m. **Lektion 4**, när kodexemplen växer, kommer vi att gå över till att spara våra program i filer och köra dem i Pythons utvecklingsmiljö **IDLE** (sid 21).

Lektion 2

Variabler

Python som programmerbar kalkylator

Python_Progr

Öppna Python, mata in följande och tryck efter varje rad på Enter:

```
>>> v = 14
>>> t = 2
>>> h = v * t - 9.8 * t**2 / 2
>>> print(h)
8.399999999999999

>>>
```

Vi får förstås samma resultat som i **Lektion 1**. Men där matade vi in *aritmetiska uttryck*. Här matas in *algebraiska uttryck* genom att införa *variabler*: **v**, **t** och **h** i koden ovan. Dessutom använder vi oss av en smartare variant av uttrycket (se fråga 5 i **Lektion 1**, sid 10).

Variabler är platshållare för värden. I datorn lagras deras värden i minnesceller. I koden använder vi oss av deras *namn* för att komma åt minnescellers *innehåll*, både för att skriva till dem och för att läsa från dem. Att ge en variabel ett värde (skriva till den) kallas för *tilldelning*. Variabler måste tilldelas värden *innan* de används. I koden ovan tilldelas **v** och **t** först värden för att sedan – i nästa rad – användas i det algebraiska uttrycket till höger om **=** . Där för att definiera **h**.

Tilldelningsoperatorn =

Tilldelningen av värden till variabler sker med **=** , den s.k. *tilldelningsoperatorn* som *inte* är identisk med matematikens likhetsteken (*Koda matte med Python*, sid 31-33). Operatorn **=** tilldelar från höger till vänster, t.ex. `v ← 14`. Här tilldelas variabeln **v** värdet 14, i koden: `v = 14`. Tilldelningsoperatorns prioritet är lägst bland operatorerna (*Koda matte med Python*, sid 22). Ett annat exempel:

```
h = v * t - 9.8 * t**2 / 2
```

Här beräknas *först* hela det algebraiska uttrycket till höger om **=** enligt ***PEMDAS***. *Sist* tilldelas resultatet variabeln **h**. Slutligen skrivs dess värde ut med hjälp av **print()**.

Kodförklaring

Python_Progr

Sista raden	**print()** är en fördefinierad funktion i Python som skriver ut till skärmen värdet på det som står i parentesen, i det här fallet värdet på **h** som har beräknats i raden innan.

Frågor

1. Skriv i Python `print(a)`. Vad händer och varför? Vad är `a` här?
2. Vilken kod t.ex. måste du lägga till *innan* du skriver `print(a)` för att det ska gå bra?
3. Vilken regel följer av experimenten i frågorna 1 och 2? Formulera regeln.
4. Vilken kod måste föregå koden `x = x + 1` för att `print(x)` sedan ska ge resultatet `651`? Skriv tre satser i Python för att verifiera resultatet.
5. Vad ger däremot `print('a')`? Vad är `'a'` här?
6. Skapa en variabel `a`. Skriv sedan kod som åtstadkommer utskriften `a = 3` .

Övningar

1201

Skapa två variabler och tilldela dem värdena `5` och `3`. Tilldela deras summa till en tredje variabel. Generera utskriften:

```
Summan av 5 och 3 är 8
```

Utskriften ska ske med hjälp av variablerna.

1202

Utveckla lösningen till övn **1201** vidare så att du får följande utskrifter efter varandra:

```
Summan av 5 och 3 är 8

Differensen 5 - 3 är 2
```

Använd samma variabler som i **1201**. Lägg till endast en till variabel för differensen. Skriv ut med hjälp av variablerna.

1203

Komplettera lösningen till övn **1202** så att du får följande utskrifter efter varandra:

```
5  +  3  ger  8
5  -  3  ger  2
5  *  3  ger  15
5  /  3  ger 1.6666666666666667
5  // 3  ger  1
```

Skapa ytterligare variabler och tilldela de fyra räknesätten till dem. Inkludera Pythons operator `//` för heltalsdivision. Skriv ut med hjälp av variablerna.

1204

Varför ger följande kod ett felmeddelande i Python?

```
a   = 1
sum = sum + a
print('\n\t sum =',  sum, '\n')
```

Hitta felets orsak. Åtgärda felet.

1205

Vilka värden kommer variablerna `tal` och `prod` att ha efter följande kod?

```
tal  = 5
prod = tal * 4
tal  = tal + tal + tal + tal
```

Svara först utan Python. Testa sedan i Interactive mode. För vilken matematisk kunskap är koden ett exempel på?

1206

Vilka värden kommer variablerna `tal` och `potens` att ha efter följande kod?

```
tal    = 5
potens = tal ** 4
tal    = tal * tal * tal * tal
```

Gör samma sak som i övn **1205** och besvara samma fråga.

1207

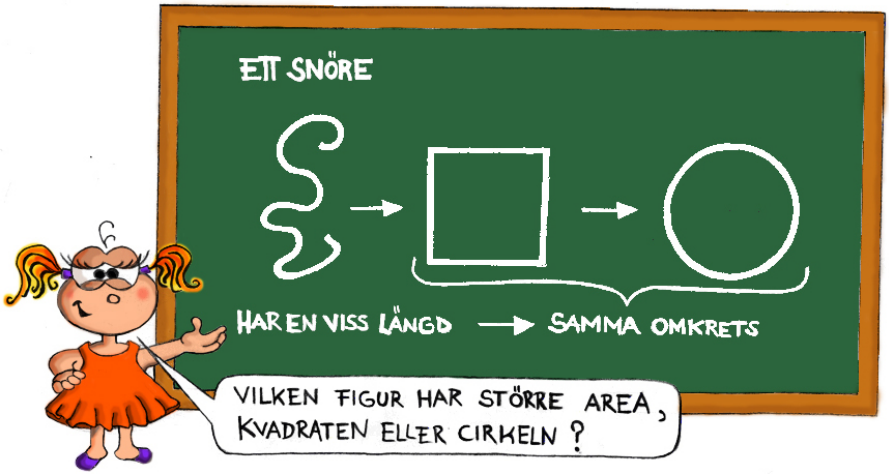
Vilka värden får `c` och `d` efter följande kod?

```
a = 123456789
b = a
c = b ** (a - b)
d = c // (b - a)
```

Svara först. Testa sedan i Python.

Lektion 3

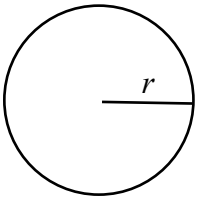
Problemlösning

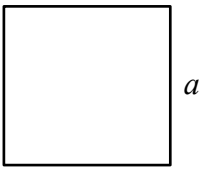


Matematisk förberedelse

Cirkel-kvadrat problemet

Vi betecknar cirkelns radie med r och kvadratens sida med a :





Cirkelns omkrets: $O_{cirkel} = 2\,\pi\,r$ Kvadratens omkrets: $O_{kvadrat} = 4\,a$

Båda figurernas omkrets ska vara lika stor, vilket innebär:

$$O_{kvadrat} = O_{cirkel}$$

$$4\,a = 2\,\pi\,r$$

$$a = \frac{2\,\pi\,r}{4}$$

Samband mellan r och a :

$$\boxed{a = \frac{\pi\,r}{2}}$$

Sambandet ovan gäller om och endast om cirkeln och kvadraten har samma omkrets. Vi tar över den till den digitala lösningen, se **blå**markerad kod nedan:

Digital lösning

Circle_Square_1

Öppna Python, mata in följande och tryck efter varje rad på Enter:

```
>>> from math import pi
>>> r = 4
>>> a = pi * r / 2
>>> A_circle = pi * r**2
>>> A_square = a**2
>>> print('Cirkelns area är', A_circle)
Cirkelns area är 50.26548245743669
>>> print('Kvadratens area är', A_square)
Kvadratens area är 39.47841760435743
>>>
```

Lösningen visar:

Cirkelns area är större än kvadratens.

I denna version av lösningen har vi besvarat frågan om cirkeln eller kvaraten får större area när de har samma omkrets endast för specialfallet cirkelns radie `r = 4`.

Men kommer resultatet vara det samma om man väljer andra värden för radien än `r = 4`? Denna version kan inte besvara frågan därför att värdet på `r` är hårdkodat i programmet. Denna version måste vidareutvecklas så att man kan testa programmet för vilken radie som helst. Detta kräver att vi *generaliserar* koden, t,ex, genom att läsa in värdet på radien. I nästa lektion ska vi lära oss att läsa in data till Python och koombinera detta med att skriva och spara koden i en fil.

Kodförklaring

Circle_Square_1

Första raden

`from math import pi` importerar konstanten π från modulen `math` till vår session så att vi kan komma åt den sedan. Konstanten `pi` är lagrad i modulen `math`. En modul är en samling fördefinierade koder i Python.

Raderna 6 & 8

`print()` skriver ut text och en variabel. Kommat skiljer åt texten från variabeln. Texten omges i koden av `' '` som är apostrofer. Variablerna `A_circle` och `A_square` däremot som är beräknade och tilldelade värden innan, står med sitt namn utan apostrofer, skilda med komma.

`if`-satsen

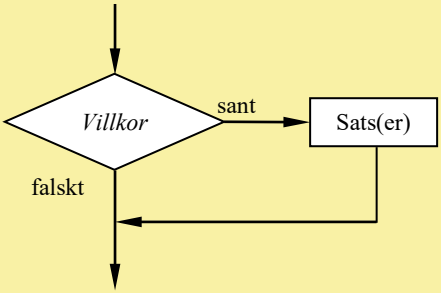
Övningar 1301
1305-1307

```
if villkor :
    Sats(er)
```

Om villkoret är sant utförs en eller flera satser. Är villkoret falskt, görs ingenting. Ett *val* mellan ett alternativ eller ingenting *(enkel selektion)*.

Indragning(ar) avgör satsernas räckvidd.

Flödesschemat visar **`if`**-satsens logik:



Frågor

1. Kör koden **`Circle_Square_1`** (sid 18) utan den första raden **`from math import pi`**. Tolka Pythons felmeddelande.
2. Varför definieras i koden **`Circle_Square_1`** kvadratens sida **`a`** till uttrycket **`pi * r / 2`** medan cirkelns radie `r` sätts godtyckligt till 4?
3. Vilket resultat får man om man kör **`Circle_Square_1`** med andra värden på `r` än 4? Testa med några i Python. Tror du att resultatet är oberoende av `r`-värdena?
4. Varför blir det fel om man i **`print`**-satsen utelämnar kommat i parentesen?
5. Sätt i **`print`**-satsen `A_circle` och `A_square` inom apostrofer och kör. Tolka resultatet.
6. Varför står i **`print`**-satsen apostrofer kring den första, men inte kring den andra *parametern* (delar i **`print()`**-satsens parentes åtskilda med komma)?

Övningar

1301

1. Ersätt i koden **Circle_Square_1** (sid 18) den första raden `from math import pi` med:

```
import math
```

2. Ersätt den tredje raden med:

```
a = math.pi * r / 2
```

3. Ersätt de två `print`-satserna var med följande `if`-sats:

```
if A_circle > A_square :
   print('Cirkeln är störst.')

if A_circle < A_square :
   print('Kvadraten är störst.')
```

Gör andra ändringar som behövs. Testa den nya varianten av **Circle_Square_1** med olika värden på `r`.

1302

Lös ut r ur sambandet mellan `r` och `a` (sid 17). Modifiera koden **Circle_Square_1** genom att ge först `a` ett värde. Använd sedan det nyvunna sambandet för att definiera `r`. Jämför resultaten för olika värden på `a` och med tidigare.

1303

Modifiera koden **Circle_Square_1** så att den besvarar frågan: Hur många hela procent är cirkelns area större än kvadratens? Kör koden för olika värden på `r`. Påverkar detta procentsatsen?

1304

Lös ***Cirkel-kvadrat*** problemet (sid 17) generellt: Ställ upp ett algebraiskt uttryck för förhållandet mellan figurernas areor, t.ex.: $A_{cirkel} \,/\, A_{kvadrat}$

a) Ange kvoten *exakt* genom att både under uträkningen och i resultatet bibehålla π som bokstav och använda bråk istället för decimaltal.

b) Vilken generell slutsats kan man dra av resultatet? Är slutsatsen beroende av figurernas storlek?

c) Ange svaret i hela procent. Jämför resultatet med Pythons svar i övn **1303**.

Lektion 4

Från kalkylator till program

Hittills har vi använt Python både som smart och programmerbar kalkylator (sid 9 & 13). I båda fall har vi kört Interactive mode vars nackdel dock är att koden är borta för gott så snart man lämnat Python. Man har inte längre kvar koden varken för att testa den igen eller för att utveckla den vidare med nya idéer som man kommer senare på. I längre program blir det här arbetssättet ohållbart. Att varje gång behöva mata in den gamla koden för att lägga till en liten förbättring, är slöseri med tiden, dessutom jätte tråkigt. I sådana fall vill man ha möjligheten att spara koden i en fil för att återanvända och vidareutveckla den.

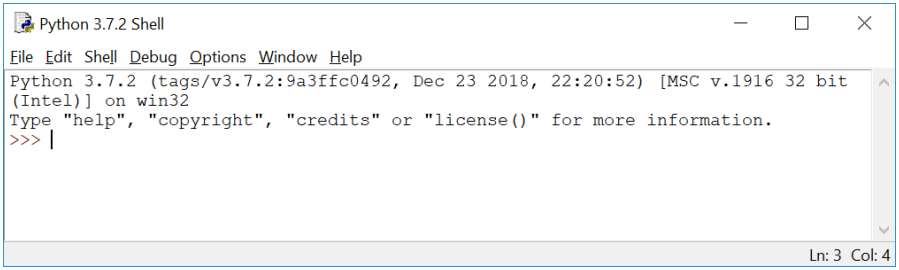
En miljö där koden kan både skrivas, sparas i en fil och köras är Pythons **IDLE** som står för *Integrated Development and Learning Environment* och automatiskt följer med när man installerar Python enligt instruktionerna i appendix **A** (sid 101).

Öppna Pythons IDLE genom att klicka på ikonen →

Den ser lite annorlunda ut än pythonikonen på sid 9

och ligger ofta bredvid den. Följande vitt fönster öppnas

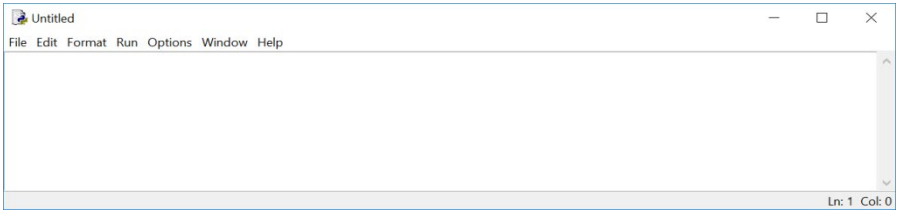
som är IDLEs *kommandofönster*, även kallat Shell:



Gå till menyraden längst upp och klicka på menyn File och sedan på

undermenyn New File: **File → New File**

Ett annat vitt, tomt fönster öppnas, IDLEs *editfönster*:



Mata in koden på nästa sida (utan radnumren) i *editfönstret* ovan:

```
1     # Circle_Square_2.py
2     # Läser in cirkelns radie, beräknar kvadratens sida och
3     # båda figurernas area samt jämför dem med varandra
4
5     import math                       # Importerar modulen math
6     r = int(input('\n\tMata in cirkelns radie:\t'))  # Inläsning
7     a = math.pi * r / 2               # Kvadratens sida, sid 17
8
9     A_circle = math.pi * r**2         # Cirkelns area
10    A_square = a**2                   # Kvadratens area
11
12    if A_circle > A_square :
13       print('\n\tCirkelns area är större än kvadratens.\n')
14    else :
15       print('\n\tKvadratens area är större än cirkelns.\n')
```

OBS! Endast av estetiska skäl har vi avbildat editfönstrets kod i en helt annan grafisk form än i det vita, lite tråkiga **IDLE**-editfönstret som visas på förra sidan. Dessutom tillämpas radnumrering för att kunna hänvisa till kod som ska förklaras, se ***Kodförklaring*** på nästa sida.

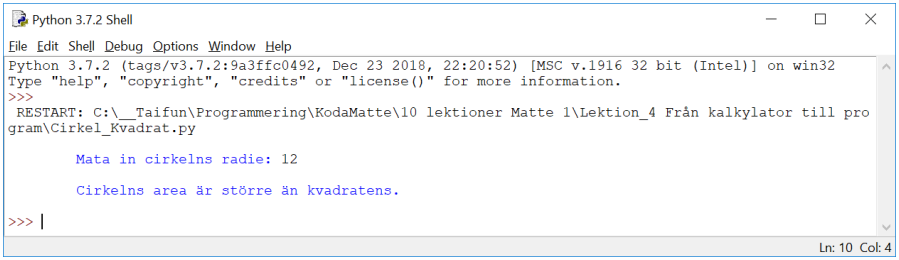
Spara koden i en fil

Välj i editfönstrets menyrad **File → Save As…**, välj lämplig plats på din dator, ange filnamnet **Circle_Square_2.py** och klicka på **Spara.**

Exekvera programmet

Välj i editfönstrets menyrad: **Run → Run Module.**

Du borde nu se följande utskrift i *kommandofönstret* efter du matat in ett värde för **r** :



Här har vi återvänt och visar igen IDLE:s *kommandofönster* i originalskick (Shell) som vi började med när vi startade IDLE (sid 21).

Kodförklaring

`Circle_Square_2`

Raderna 1-3, 5, 7, 9-10

Tecknet **#** inleder i Python en *radkommentar*. Kommentarer utförs inte av Python utan ska endast förklara koden.

Rad 6

Koden **input(...)** läser in användarens inmatning. Koden **int(...)** omvandlar det inlästa till ett heltal som tilldelas variabeln **r**, cirkelns radie.

Raderna 6, 13, 15

Koden **\n** (**n**ewline) genererar radbyte i utskriften.
\t (**t**abulator) skapar horisontellt avstånd.

På Mac-datorer kan man få tecknet backslash (****) genom tangentkombinationen <alt>-<shift>-</>, dvs **/** utan att släppa tangenterna <alt> och <shift>.

Tecknen **'** **'** är apostrofer. I pythonkod måste text alltid omges av apostrofer. Själva tecknet **'** är endast kod och kommer inte att visas i utskriften.

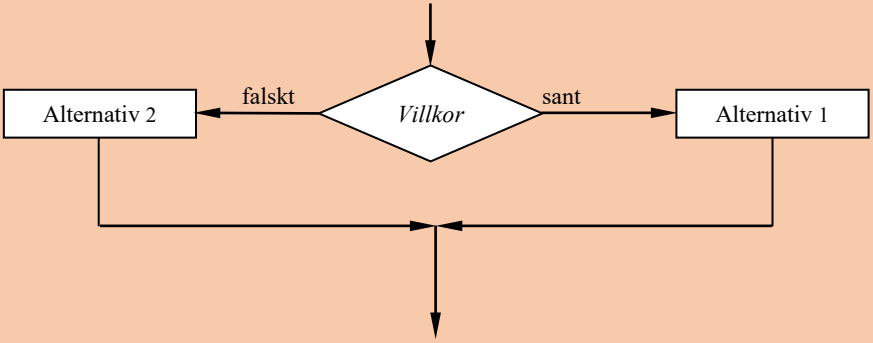
Raderna 12-15

```
if villkor :
    Alternativ 1
else :
    Alternativ 2
```

Ett s.k. *tvåvägsval* mellan två alternativ som utesluter varandra. Beroende på om *villkoret* är sant eller falskt utförs ett av alternativen.

Indragningarna avgör alternativens räckvidd. (**raderna 13, 15**).

Flödesschemat visar **if-else**-satsens logik:



Frågor

1. Vad är skillnaden mellan att köra Python i Interactive mode och att använda **IDLE**?
2. Vad är skillnaden mellan **IDLE**:s kommando- och editfönster? I vilket skrivs koden?
3. Går det i **IDLE** att skriva pythonkod och exekvera den utan att spara den i en fil? Om ja, gör det. Om nej, förklara när det inte går.
4. Kan man skriva om i programmet **`Circle_Square_2`** (sid 22) satsen i rad **`6`** som läser in cirkelns radie, till två separata satser? Om ja, vilka satser blir det då?
5. Varför blir del fel om man i rad **`6`** av **`Circle_Square_2`** tar bort koden **`int(...)`**?
6. Är det möjligt att i raderna **`12-15`** av programmet **`Circle_Square_2`** ersätta **`if-else`**-satsen med två **`if`**-satser? Om ja, gör det.

Övningar

1401

Öppna Pythons **IDLE** eller din favorit utvecklingsmiljö, mata in koden till programmet **`Circle_Square_2`** (sid 22), spara och exekvera den. Kör flera gånger med olika värden på cirkelns radie.

Har du ingen favorit pythonmiljö, ladda ner och installera Python enligt appendix **A** (sid 101). Då följer **IDLE** med. Det går på 5 min. Skriv, spara och exekvera **`Circle_Square_2`** enligt instruktionerna på sid 21.

1402

Skriv ett pythonprogram i en fil så att du får följande utskrift:

```
Välkommen till
Koda matte med Python!
Programmering i matematik.
```

Spara filen och exekvera programmet.

1403

Det duala ***Cirkel-kvadrat***-problemet:

En cirkel och en kvadrat har *samma area*. Vilken av dem har den *minsta omkretsen*?

a) Ställ upp ett samband mellan cirkelns radie r och kvadratens sida a. Använd sambandet i ett pythonprogram liknande **`Circle_Square_2`** (sid 22). Läs in ett exempel, beräkna figurernas omkrets och jämför dem med varandra. Kör programmet för andra exempel. Besvara frågan ovan.

b) Lös problemet generellt: Ställ upp ett algebraiskt uttryck för förhållandet mellan figurernas omkretsar:

$$O_{cirkel} \;/\; O_{kvadrat}$$

Svara *exakt*, se övn **1304**

c) Beräkna den procentuella skillnaden mellan figurernas omkretsar.

Appendix

A. Installation av Python

På PC

Öppna din webbläsare, gå till adressen:

www.python.org/downloads

Klicka på knappen **Download Python x.x.x**

Installationsfilen *.exe laddas ner. Dubbelklicka på den. Dialogrutan **Install Python…** dyker upp. Bocka först för:

Add Python x.x to PATH

Klicka sedan på **Install Now**. För att öppna Python interpretatorn:

Klicka på den lilla pythonikonen  från datorns Start-knapp.

Mata in pythonkod enligt nästa sida.

På Mac

Öppna din webbläsare, gå till adressen www.python.org/downloads

Klicka på knappen **Download Python x.x.x** . Filen *.pkg laddas ner.

Dubbelklicka på den. Följ instruktionerna.

Efter installationen öppnas mappen **Python x.x** där du ser ett antal filer. En av dem heter **IDLE.app**. Dubbelklicka på den.

Pythons programmeringsmiljö IDLE öppnas. Mata in kod enligt nästa sida.

Python interpretatorn

Det du ser efter prompten **>>>** där markören står och blinkar, kallas för *Python interpretatorn*. Eftersom Python är ett interpreterande språk kan vi använda s.k. *Interactive mode*. Dvs vi kan mata in pythonkod i interpretatorn och få svar direkt. Mata in t.ex. den rödmarkerade koden i Python interpretatorn som du redan öppnat. På Mac-datorer kan man få tecknet backslash (****) genom tangentkombinationen <alt>-<shift>-<**/**>, dvs **/** utan att släppa tangenterna <alt> och <shift>.

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> print('\n\t Välkommen till Koda matte med Python! \n')

         Välkommen till Koda matte med Python!

>>> exit()
```

Pythons svar är den svarta texten under den röda. Dvs Python skriver ut den text som du innan bakat in i **`print()`**-satsens parentes (i rött).

Lämna Python med `exit()`.

Pythons IDLE

En beskrivning av denna pythonmiljö *(**I**ntegrated **D**evelopment and **L**earning **E**nvironment)* finns på sid 21 där vi första gången skrivit pythonkod i en fil, sparat och exekverat den i en och samma miljö. Programvaran följer automatiskt med när man installerar Python enligt instruktionerna ovan.

B. Tillägg av grafisk miljö

På PC

Öppna Kommandotolken på din PC, ett svart fönster med rubriken:

Command Prompt

Mata in följande kommando:

```
pip install matplotlib
```

Vänta tills du får tillbaka prompten.

På Mac

Öppna Terminal på din Mac, ett vitt textfönster med rubriken:

Terminal

Mata in följande kommando:

```
pip3 install matplotlib
```

Vänta tills du får tillbaka prompten.

pip står för "pip installs packages" (en s.k. rekursiv akronym) är en hanterare av tilläggsprogramvaror för Python. Den följer automatiskt med när vi installerar Python enligt appendix **A**. **pip3** syftar åt version **3**. Med kommandot ovan har vi laddat ner och installerat det externa biblioteket **matplotlib** vars moduler inte finns med i pythonmiljöns standarduppsättning (Appendix **A** och **B**).

Matplotlib

Det här biblioteket innehåller i sin tur moduler skrivna i Python som med få rader kod kan generera grafiska miljöer för ritning av 2D figurer, diagram, grafer till matematiska funktioner och andra grafiska objekt. Den viktigaste av dem är modulen **pyplot** som kan importeras med satsen:

```
import matplotlib.pyplot as plt
```

Efter den har man tillgång till matplotlibs alla grafiska miljöer. Syntaxen liknar MATLAB-kod som är ett avancerat numeriskt matematikprogram med dyr licens.

Om programmering

Programmering är i allra högsta grad ett praktiskt ämne. Ingen kan lära sig programmering genom att bara läsa böcker: *Learning by doing!* Man måste själv skriva och testa program för att lära sig programmering. För att kunna göra det behövs ett verktyg som i sin tur är en programvara – en s.k. *kompilator* eller *interpretator.* Medan en kompilator *översätter* källkod (t.ex. Python) till maskinkod (datorns språk) och lagrar den i en exekverbar fil, *tolkar* en interpretator källkoden direkt till maskinkod och exekverar den direkt. Människan kan lära sig, förstå och skriva källkod, men inte maskinkod. Datorn arbetar tvärtom. Jobbet däremellan görs av de ovannämnda verktygen. Det finns hundratals språk för källkoden. Men språket är endast ett medel av underordnad betydelse. Målet är att lära sig *tankesättet* och *tekniken* att programmera, oberoende av språk. Har man en gång förstått de grundläggande koncept som är gemensamma för alla språk, blir det närmast en teknikalitet att på egen hand lära sig ett nytt språk. Vi har bestämt oss för programmeringsspråket *Python* för att kunna praktisera och lära oss programmering.

Programmeringsspråket Python

Python skapades år 1989 av Guido van Rossum, en forskare på *National Research Institute for Mathematics and Computer Science* i Amsterdam. Python är ett *interpreterande* programmeringsspråk, dessutom ett *universellt* språk, dvs det lämpar sig för alla möjliga tillämpningar – små som stora, enkla som komplexa. Python kan enkelt, snabbt och gratis installeras på alla datorplattformar utan att man behöver bry sig om licenser. Koden är enkel, elegant och självbeskrivande, ligger nära pseudokod och återspeglar på ett intuitivt sätt *algoritmers* struktur.

Programmeringsmiljöer

För att kunna köra Python måste vi antingen installera Python lokalt på vår dator enligt appendix **A** eller använda en redan befintlig pythonmiljö på Internet.

Vi rekommenderar det första: Du kommer att ha stor nytta av det när du i fortsättningen vill organisera dina pythonprogram i filer och gå vidare med programmering. T.ex. när du skriver *funktioner* där du skapar dina egna separata moduler för att återanvända befintlig kod (Lego-principen). Dessutom kommer du att alltid ha tillgång till Python på din dator oberoende av Internet.

Om du däremot vill slippa installation av programvara, öppna din webbläsare på datorn eller på mobilen. Gå till appen *Mattekollen* med adressen app.mattekollen.se. Välj alternativet En mobil pythonmiljö och kör Python. Inget konto krävs. Andra webbaserade utvecklingsmiljöer är bl.a. repl.it (konto behövs) och colaboratory.