Allmänna instruktioner

Välkomna till laborationer i nätverksprogrammering!

Här följer de regler som gäller för samtliga labbar när annat ej anges.

Frivilligt eller obligatoriskt?

De flesta labbar är obligatoriska men några är frivilliga. Av *laborationens rubrik* bör det framgå om den är obligatorisk eller frivillig. Obligatoriska labbar har ingen deadline. Frivilliga labbar har en deadline. Om man gör dem frivilliga labbarna före deadline, så kan de ge labpoäng. Om man bli godkänd i aprils tentan, så ränkas eventuella labpoäng som en poängbonus på tentan (dvs. man kan på detta sätt höga betyget).

Jobba individuellt

Uppgifterna skall utföras individuellt! Med det menas att man själv skall sitta bakom tangentbordet och knappa in all kod. Man *får* rådfråga sina kamrater, och labbansvarig, men man *får inte* kopiera någon annan persons kod och *inte heller* låta någon annan person kopiera ens egen kod. *Du får inte göra din lösning tillgänglig för andra!*

Redovisning

Redovisa dels genom att ladda upp din källkodsfil eller pdf till pingpong. *För att få s.k. labpoäng måste uppladdning ske innan deadline.* Därtill måste man också redovisa muntligt genom att demonstrera programmet för labbansvarig och förklara sin kod.

Närvaroplikt

Eftersom labbarna skall kunna redovisas muntligt är det (i princip) obligatoriskt att närvara under laborationstillfällena och att arbeta aktivt med labbarna!
För att hinna utföra labbarna måste man också arbeta utanför labtid.

Labpoäng

De frivilliga labbarna ger s.k. labbpoäng som senare omräknas till tentamenspoäng.

Hur många poäng man kan få för en uppgift framgår av pingponginlämningen. Självklart får man avdrag för buggar och ful kod.

Lab 1

[Obligatorisk] Python: Introduktion

Målet med denna laboration är att man skall lära sig grunderna i pythonprogrammering. Laborationen är relativt omfattande. Vid sidan av schemalagd labtid måste man arbeta med laborationen hemma på egen hand!

1.0 Inledning

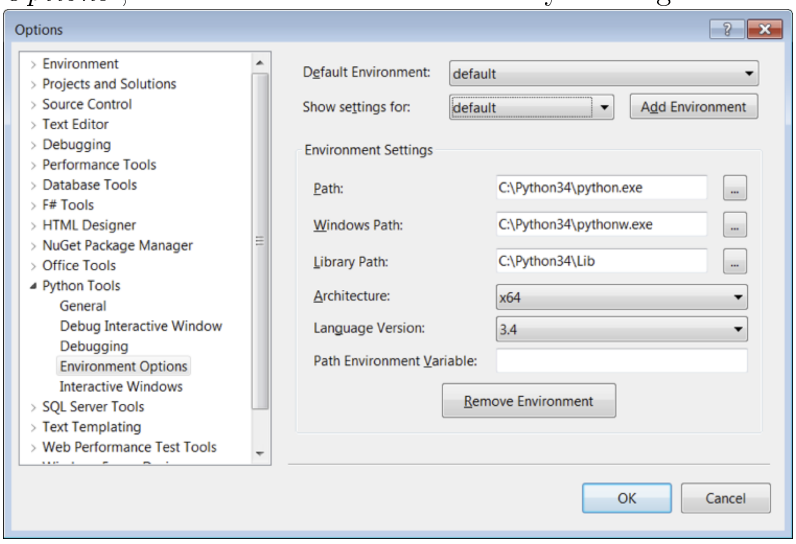
Jag (Ragnar) kan inte skriva en bättre tutorial än den som redan finns på pythons officiella webbsajt! Den aktuella handledningen innehåller därför bara lite inledande instruktioner som hjälper er att komma igång och hitta fram till detta webbdokument!
Därutöver innehåller handledningen läsförståelsefrågor på kapitel 1-7 i webbdokumentet. Det är meningen att man *först* skall läsa och knappa sig igenom ett kapitel i webbdokumentet (och testa alla kommandon etc. som finns beskrivna där) och *därefter* gå till läsförståelsefrågorna. Om man gör dessa saker i omvänd ordning kommer man att lära sig *mindre*, eftersom läsförståelsefrågorna inte täcker hela webbdokumentet!

Redovisning

Man behöver inte redovisa denna laboration, men alla pythonkunskaper är viktiga att ha i kursens övriga laborationer och på tentan.

1.0.1 Installera python

Skolan har förhoppningsvis installerat python 3.x i sina salar (version 3.6.4 borde det aktuellt vara). Det kan hända att Visual Studio tycks ha python men ändå ställer till problem då du försöker köra *helloworld*-programmet. Öppna då *TOOLS->Options* och välj *Python Tools -> Environmnet Options* , klicka *Add Environment* och fyll i enligt nedan



Om Visual studio saknar python bör du från skoldatorn kunna installera det via Software Center.
Du kan också installera python på din egen **privata** dator. Detta är gratis och mycket enkelt. Surfa till **python.org**, och ladda ned python via *downloads* (eller installera Visual Studio 2017, och installera Python via det programmet). Det är *version 3.x* (eller senare) som gäller! Mac-datorer har redan en version av Python installerad, och om det är Python 2.x, behöver du installera Python 3.x (aktuellt version 3.6.4).

Om du har Windows och Visual Studio 2015 på din privata dator bör du också installera ett plugin som gör det möjligt att skapa och debugga pythonprojekt i Visual Studio 2015. Googla på *python visual studio 2015* så hittar du länken till *microsoft.github.io/PTVS*. Installera detta på din privata dator!

1.0.2 Öppna ett kommandofönster och testa

I denna inledande övning skall du öppna ett kommandofönster (gör detta även om du har tillgång till visual studio):

1. Öppna ett kommandofönster (i Windows söker man på *Windows Powershell* i start-menyn och på Macen skriver man *terminal* i spotlight)!
2. Skriv kommandot `python3` (och tryck enter) i terminalfönstret. Om det inte fungerar så prova kommandot `python` utan 3:an.
   (a) Förhoppningsvis ser du nu en pythonprompt (`>>>`), vilket innebär att du är inne i pythons interpretator (kommandotolk).
   (b) Skriv `3+2` och tryck return. Verifiera att interpretatorn skriver ut talet `5`!
   (c) Ge kommandot `print("Hello, world!")` , och verifiera att interpretatorn skriver ut texten *Hello, world!*
   (d) Ge kommandot `quit()`, och verifiera att du nu lämnar python och återigen hamnar i kommandofönstret.
3. Öppna en lämplig texteditor (t.ex. notepad++ i windows)
   (a) Skapa en ny fil. Skriv in texten `print("Hello, world")`.
   (b) Spara filen under namnet *hello.py* i lämplig katalog.
4. Åter i kommandofönstret. Navigera[1] till ovanstående katalog där du sparat `hello.py`
5. Skriv kommandot `python3 hello.py` (skippa eventuellt 3:an) och verifiera att ditt program körs och skriver ut *Hello, World!* på skärmen!

1.0.3 Testa från Visual Studio

Om du sitter vid en Windowsdator med Visual Studio och python bör du absolut använda visual studio:

1. Starta Visual Studio
2. Skapa ett nytt projekt med valfritt namn. Ange att du skall skapa ett python projekt (inte iron python, etc)
3. Du bör ha fått en fil med filändelsen `py`. Skriv in texten `print("Hello, world!")`

[1] Några kan eventuellt sakna vana att använda kommandofönstret. Skriv `cd` och därefter sökvägen till den aktuella katalogen. Istället för att skriva in sökvägen kan man med musen dra den aktuella katalogen från utforskaren/finder och släppa den i kommandofönstret.

4. Tryck på RUN-knappen (grön trekant) och provkör, verifiera att det fungerar!

5. Som pythonprogramerare vill man *alltid* ha en interaktiv kommandotolk tillgänglig, så att man kan provköra och experimentera med alla satser man känner sig osäker på –innan man skriver in dem i sitt program.

   (a) Från visual Studios *TOOLS meny*, välj *Python* och sedan *Python Interactive Window*.

   (b) Verifiera att ett fönster med en pythontolk blir synligt.

   (c) Provkör kommandot `3+2` och verifiera att resultatet blir `5`.

1.0.4 Webbdokumentets tutorial

På *python.org* finns det en tutorial för att lära sig pythons grunder.

Besök *python.org*. Hovra över *Documentations*, välj *Python 3.x Docs*. Klicka på länken **Tutorial** *start here*. Länken leder till tutorial som *webbdokument*.

Part 1 till 9 av denna tutorial är viktig och ingår i kursen. Den här labben täcker cirka part 1 till 7.

Det som återstår av denna labhandledning är läsanvisningar och kontrollfrågor. Kapitelnumreringen i återstoden av labhandledning överensstämmer med numreringen i det aktuella webbdokumentet.

1.1 Whetting Your Appetite

Detta är ett kort kapitel som innehåller lite reklam för programspråket python. Läs gärna!

1.2 Using the Python Interpreter

Kapitlet innehåller en handledning i hur man startar pythons kommandotolk. Vi har redan gjort detta i kapitel 0, men här finns ytterligare lite information. Läs gärna!

1.3 Introduktion

Detta kapitel innehåller en introduktion till python. Bland annat behandlas här:

- Nummer och operationer på dessa
- Strängar och några strängoperationer
- Listor, som påminner om arrayer, stackar, fifoköer, etc.
- Programmering. While-loopar utan krullparanteser etc.

Läs kapitlet noga! Knappa in och provkör alla exempel i det interaktiva pythonfönstret i visual studio (eller i annan pythontolk). Om Visual Studio fungerar som den skall får man *interaktiv intellisense-hjälp i kommandofönstret!*

1.3.1 Kontrollfrågor på Numbers

1. Om man dividerar två heltal med /-operatorn. Blir resultatet då alltid ett heltal (som det blir i programspråket C)?
2. Hur utför man heltalsdivision i python?
3. Vad är `2**-1` ?

1.3.2 Kontrollfrågor på Strings

1. Vad blir `15*"tjata "` ?
2. Vad är enklaste sättet att skriva in en sträng bestående av flera rader?
3. Hur skapar man enklast en sträng som innehåller "citattecken"?
4. Antag att `s = "Hello, world!"`
   (a) Vad blir då `s[0:5]` ?
   (b) Vad blir då `s[:5]` ?
   (c) Vad blir då `s[-1]` ?
5. Hur tar man reda på längden av en sträng?
6. Provocera fram en felutskrift som visar att python strängar inte kan muteras!

1.3.3 Kontrollfrågor på Lists

1. Antag att man gör följande

```
a  = [100,1,2,3,4,5,6,7]
r = a
s =  a[0:3]
t = a[:]

a[0]=99
```

   (a) Vad blir då `r` ?

   (b) Vad blir då `s` ?

   (c) Vad blir då `t` ?

   (d) I satsen `x = y`, kommer `y` att *kopieras* eller kommer `x` och `y` att referera till *samma objekt?*

2. Hur tar man reda på längden av en lista?

3. Vad blir `m[1][0]` om `m=[[1, 2], [3, 4]]`?

1.3.4 Kontrollfrågor på First steps...

1. Antag att `a` och `b` är två variabler. Vad gör nedanstående sats?

```
a,b   =   b,a
```

2. Skriv en `while`-loop som skriver ut alla heltal mellan 1 och 9 på skärmen!

1.4 More Control Flow Tools

Läs detta trots att du redan kan programmera i andra språk. Det kan ju skilja sig lite...

1.4.1 Kontrollfrågor på if satsen

1. Varför, tror du, använder man `elif` istället för `else if` (som i C)?

1.4.2 for-satsen

1. Antag att `djur=["hund, "katt", "elefant"]`. Skriv en `for`-sats som skriver ut alla element i `djur`!

2. Nedanstående kod är hämtat från webbdokumentet

```
for w in words[:]:
    if len(w) > 6:
       words.insert(0, w)
```

   Av vilken anledning loopar man över `words[:]` och inte `words` ?

1.4.3 range() funktionen

1. Skriv med hjälp av `range()`funktionen en `for`-loop som skriver ut alla heltal mellan 0 och 99!

2. Man skulle kunna tro att `range()`-funktionen returnerade en lista, men så är inte fallet. Skapa med hjälp av `range-` och `list-` funktionerna en lista som innehåller alla heltal mellan 0 och 99!

1.4.4 break, continue och else

1. En `for`-loop kan ha en `else`-gren. När utförs den?

1.4.5 Kontrollfrågor tom sats

1. I C-program ser man ibland tomma satser i form av ensamma semikolon eller tomma krullparantesblock (`;` eller `{ }`). I python använder man varken semikolon eller krullparanteser. Hur skriver man en tom sats?

1.4.6 Defining functions

1. Skriv en funktion med namn `hello` som tar ett heltal `n` som inparameter och som skriver ut strängen *pappegoja* `n` gånger!

2. Vad blir `f` om man skriver `f = hello`

3. Ett C program lagrar globala data på *heapen* och lokala variabler på *stacken*. Hur gör ett python program?

4. Hur kommer man åt en global variabel från en funktion?

5. Antag att `djur=["hund, "katt", "elefant"]`. Hur kan man lägga till strängen `"loppa"` längst bak i denna lista?

1.4.7 More on defining functions

Detta är ett relativt långt kapitel som innehåller mycket information.

keyword arguments

Skriv en funktion med namn `print_all` som skriver ut alla parametrar den anropas med. Den skall kunna anropas med godtyckligt många parametrar.

lambda funktioner

1. Antag att man skriver

```
f = lambda a,b: a+b
```

   Vad blir då `f(3,2)`?

document string

1. Ungefär vad bör hända om man skriver

```
print(  print.__doc__ )
```

2. Hur sätter man en funktions document string?

1.4.8 Intermezzo: Coding Style

1. Ett pythonprogram måste indenteras korekt. Bör man indentera med tabbar eller mellanslag?

2. Vilket ASCII-tecken (tab eller mellanslag) tecken skriver Visual Studio ut då man trycker på TAB-tangenten?

1.5 Datastructures

Många viktiga datastrukturer finns inbyggda i python, exempelvis:

- Listor, som bland annat innehåller stack-operationer
- Tuples och sekvenser, som är som listor man inte kan ändra.
- Mängder
- Dictionaries, som lagrar par bestående av nyckel och data.

Det är dessa datastrukturer som gör python till ett kraftfullt språk.

1.5.1 More on Lists

1. Antag att `a = [1,2,3,1,1,1,5,7,2]` är en lista
   (a) ge ett kommando som räknar antalet 1:or i listan.
   (b) Kommer listan att förändras om man ger kommandot `a.sort()` eller kommer sort-funktionen att returnera en ny lista?

using lists as stacks

Vad kommer variabeln stack att innehålla efter det att följande kod har exekverats?

```
stack = [ 1, 2, 3, 4 ]
stack.append(8)
stack.pop()
stack.pop()
```

using lists as queues

Vad kommer variabeln queue att innehålla eter det att följande kod har exekverats?

```
queue = [ 1, 2, 3, 4 ]
queue.append(8)
queue.append(9)
queue.pop(0)
queue.pop(0)
```

Varför är det **dumt** att använda en Lista som en fifo-kö?

list comprehensions

Antag att `a = [0,1,2,3,4,5,6,7,8,9,10]` är en lista. Skapa med hjälp av *list comprehension* en ny lista med tvåpotenserna 1,2,4,8...

1.5.2 Kontrollfrågor på Tuples and Sequences

Låt `t = 1111, 2222, 3333` vara en tuple.

1. Vilka av följande funktioner borde man *inte* kunna anropa på `t`, och varför?
   (a) t.count(1111)
   (b) t.sort()
   (c) t.append(4444)
   (d) t.pop(0)
2. Vad gör satsen `x,y,z = t` om `t` är som ovan?

1.5.3 Kontrollfrågor på Sets

1. Antag att `answ` innehåller en sträng som en användare just knappat in. Skriv om nedanstående C++ kod

```
if (answ=="yes" || answ=="ja" || answ=="japp" || answ=="ok")
     cout << "du har svarat ja" << endl;
```

   till pythonkod som utför testet med en mängd istället för med en massa or-satser.
2. Skriv en pythhonsats som omvandlar en sträng till den mängd tecken som ingår i strängen!
3. Skriv en pythonsats som omvandlar en lista till den mängd element som ingår i listan!

1.5.4 Looping Techniques

1. Skapa en lista och skriv sedan en loop som skriver ut alla element i listan
2. Skapa ett dictionaryobjekt och skriv sedan en loop som skriver ut alla nyckel/data-par i detta objekt

1.5.5 Kontrollfrågor på More on Conditions

1. Översätt nedanstående kodfragment från C++ till python

```
if (20 <= age && age <=65)
     cout << "vad jobbar du med?" << endl;
```

   utan att använda någon och-operator.

2. Finns det en (logisk) och-operator i python och hur ser den isåfall ut?

1.5.6 Kontrollfrågor på Comparing Sequences...

1. Vilka av nedanstående villkor är sanna?

   (a) (1,2,3) < (1,3,2)

   (b) (2,1,3) < (1,2,3)

   (c) (1,2,3) < (1,2,3,4)

   (d) (2,2,2) < (1,1,1,1)

   (e) "c++"< python"

1.6 Modules

Detta kapitel handlar om hur man kan dela upp ett pythonprogram i flera filer.

Antag att filen `prog.py` bland annat innehåller en funktion `getLotto()`.

1. Hur skall man skriva för att anropa ovanstående funktion `getLotto()` från kommandofönstret?

liten övning

Skapa en fil `prog.py` men en funktion `getLotto()` som skall returnera en mängd med 7 olika tal mellan 1 och 35. Anropa sedan `getLotto()` från kommandofönstret.
Tips:

1. Låt en while-loop addera slumptal till en mängd så länge som mängdens storlek är mindre än 7.

2. importera `random` och anropa `random.randint(1,35)` för att få lämpliga slumptal.

3. I ditt kommandofönster bör du bland annat importera modulen `imp`, och göra `imp.reload()` varje gång du måste ladda om `prog.py`

1.7 Input and Output

1.7.1 Kontrollfrågor på Fancier output formatting

1. Vad returneras från anropet `"A={} och B={}".format(1,2)` ?

1.7.2 Kontrollfrågor Reading and Writing Files

1. Skriv en programsnutt som öppnar en textfil för läsning och skriver ut alla dess rader på skärmen.