Computer Programming I

5 credits

Teacher: Johan Öfverstedt
E-post: johan.ofverstedt@it.uu.se
+ A large number of teaching assistants

Lecture 1: Course information and introduction to programming with Python

The course consists of

- (Normally) 6 lectures
- A large number of lab sessions (with 5 mandatory assignments) (3 hp)
- Take home exam [during Covid-19, otherwise a normal written exam] (2 hp)

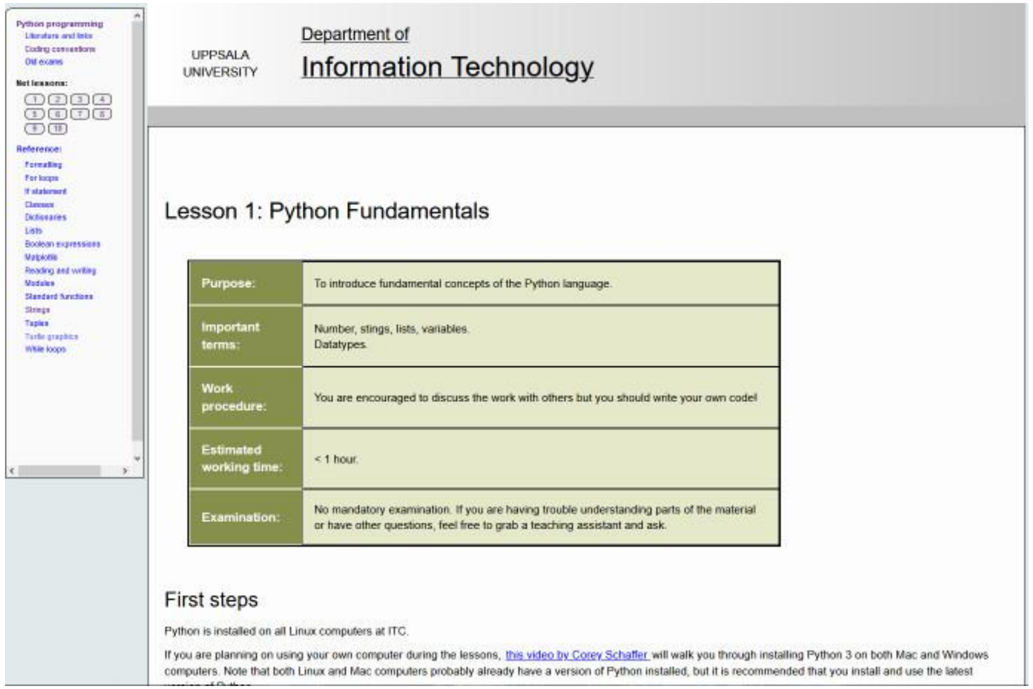
The lab sessions with individual practical work, at your computers, is essential for active learning and learning through repetition.

The lectures provide increased depth and theory to complement the assignments.

The course website

http://www.it.uu.se/edu/course/homepage/prog1/english/

The course website includes 10 web-based lessons which constitute the practical part of the course, with a set of mini-lessons to be regarded as an extra resource.



Lesson 1: Python Fundamentals

Purpose:	To introduce fundamental concepts of the Python language.
Important terms:	Number, stings, lists, variables. Datatypes.
Work procedure:	You are encouraged to discuss the work with others but you should write your own code!
Estimated working time:	< 1 hour.
Examination:	No mandatory examination. If you are having trouble understanding parts of the material or have other questions, feel free to grab a teaching assistant and ask.

First steps

Python is installed on all Linux computers at ITC.

If you are planning on using your own computer during the lessons, this video by Corey Schafter will walk you through installing Python 3 on both Mac and Windows computers. Note that both Linux and Mac computers probably already have a version of Python installed, but it is recommended that you install and use the latest

10 web-based lessons

MA = Mandatory assignment

Lesson 1: Python Fundamentals

Lesson 2: Development environment IDLE

Lesson 3: The World of Turtles

Lesson 4: Writing functions

Lesson 5: More about functions, objects and methods (MA1)

Lesson 6: Lists and tuples (MA2)

Lesson 7: Working with text (MA3)

Lesson 8: Introduction to classes

Lesson 9: Data management (MA4)

Lesson 10: Traffic simulation (MA5)

Important dates

MA = Mandatory assignment

MA1: Laborationstillfälle 8 - 15/9
MA2: Laborationstillfälle 12 - 18/9
MA3: Laborationstillfälle 15 - 24/9
MA4: Laborationstillfälle 20 - 2/10
MA5: Laborationstillfälle 28 - 14/10

It is allowed to present the assignments before the deadline.

Exam: 23/10 (Remember that you must sign up for the exam in advance to be eligible to take it)

Lab sessions

The lab sessions (which will take place remotely over the internet) will work as follows: To ask for help, advice or the ability to discuss your code with a TA (or sometimes me), you should add your name to a shared Google-document, which will be shared soon, where you write the following information at the bottom of the list:

Name, Zoom-link

You will create your own Zoom-meeting and add your link at the bottom of the list/queue. When your turn comes up, a TA will connect to your Zoom-meeting and help you out, or allow you to present your mandatory assignment orally.

Programming

Programming is a process where a set of instructions is put together to describe a computational problem and/or its solution in a formal language which can be interpreted and executed by a computing machine. There are various kinds of programming paradigms.

Functional programming: A program is a function (like in mathematics) where evaluating the function is executing the program.
Procedural programming: A program is a sequence of operations, processed in order, where each operation may read/create/update the state of the program or cause an effect to happen such as print to the screen.

Programming

Programming is a process where a set of instructions is put together to describe a computational problem and/or its solution in a formal language which can be interpreted and executed by a computing machine. There are various kinds of programming paradigms.

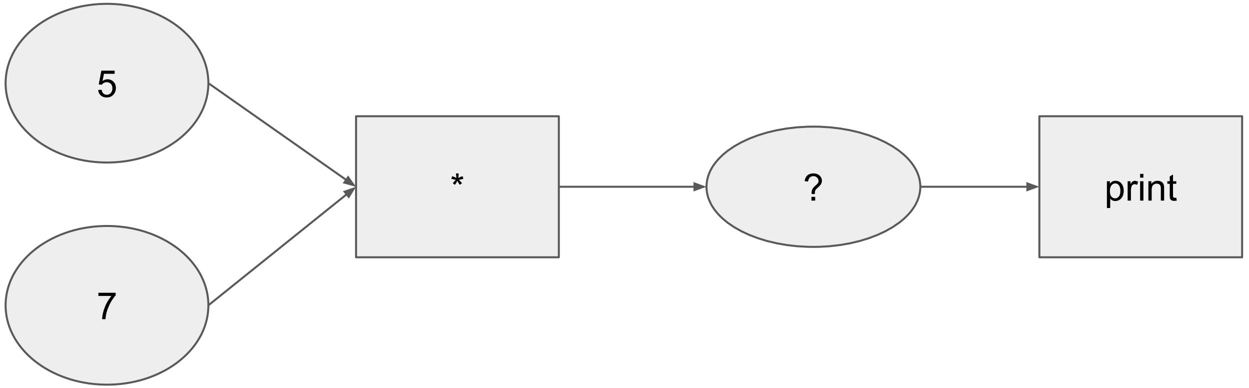
Declarative programming: A program is written as a problem/logic description and the language finds a solution without requiring specification of the solution.
Object-oriented programming: A compatible paradigm (with the others) where concepts are collected in classes/objects that contain logic/functions/procedures and data that can then be used as basic higher-level building blocks.

Programming can take on different forms

Text-based languages where the logic and program behavior is described as a textual code:

```
x = 5 * 7
print(x)
y = x + 3
print(y)
```

Programming can take on different forms



Visual node-based programming

Python

Python is a dynamically typed general-purpose programming language (often considered a scripting language) that can run programs without pre-generation of native machine-code.

Terminology

Machine-code: A binary code describing instructions of data that a computer processor can read and execute in a well-defined manner.
Compilation: A process of transforming code from a higher abstraction level into for example machine-code (or byte-code which is somewhere in between).

Interactive Python in the Terminal

Python can be written and executed interactively in a terminal, line by line, which can be a convenient way of quick experimentation. This is useful when starting to learn programming, and when you need to quickly test out a small use-case.

```
C:\Users\johan>python3
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64
bit (AMD64)] :: Anaconda, Inc. on win32

Type "help", "copyright", "credits" or "license" for more
information.
```

IDE (Integrated Development Environment)

There are a large number of IDEs that can simplify the writing, reading, and testing of Python code.

- IDLE : Included with Python
- Visual Studio Code : A free open-source IDE from Microsoft
- PyCharm : A commersial IDE with many powerful features (academic licenses can be obtained)

Anaconda: A Python-distribution

Anaconda (https://www.anaconda.com/distribution/)
Is a distribution of Python that contains a lot of popular packages that help with scientific computing, plotting, graphics, numerical computation (linear algebra), optimization, and much more.

Examples of Python-code (some arithmetic)

```
5*2**4
>> 80
```

```
5*(2**4)
>> 80
```

```
1+1*2+3
>> 6
```

The arithmetic in Python follows the usual priority rules (multiplication before addition etc).

Logic operators in Python

`a == b` : test for equality of value

`not (a)` : negation of expression

`a is b` : test for equality (identity)

`a < b, a <= b, a > b, a >= b` : inequalities

Arithmetic operators in Python

`a + b` : addition

`a - b` : subtraction

`a * b` : multiplication

`a / b` : floating-point division

`a // b` : integer division

`a ** b` : a raise to the power of b

`a % b` : modulo / remainder of division, `5 % 3 == 2`

Variables: Association of a name and a value

```
a = 2
b = 7
c = a**b
print(c)
>> 128

print(type(c))
>> <class 'int'>
```

c contains an integer.

```
a = 2.0
b = 7
c = a**b
print(c)
>> 128.0

print(type(c))
>> <class 'float'>
```

c contains a floating-point number (float).

Comments

```
# This is a comment. It will be ignored by Python.

'''
This is
a multiline
comment.
'''
```

Comments are very important for documentation of what code does, how it does it, description of assumptions, and more. Obvious facts, known to all compentent Python programmers should not be included in comments.

Strings - A first example

```python
name = 'Johan'
s = f'Hi, my name is {name}.'
print(s)
```

```
>> Hi, my name is Johan.
```

A string is a sequence of characters forming a chunk of text. Each character is represented by one or more integer-values given by a standard code (Utf-8). The f-string syntax allows creation of new strings from other strings and values.

More examples with strings

You can also create strings with the following syntax:
s = 'Hi, my name is %s' % 'Johan'

The problem with this syntax is that it is easier to make mistakes
s = 'a = %d, b = %d, c = %d' % (a, b)

```
>>
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: not enough arguments for format string
```

Writing to the screen with “print”

The procedure print is used to write to the screen / the terminal.

```
print('a', 'b', 'c')
>> a b c

print('a', 'b', 'c', sep=', ')
>> a, b, c

print('a', end=', '); print('b', end=', '); print('c')
>> a, b, c
```

Conditional statements (if)

```
a = 7
if a > 0:
  print('a is positive')
elif a == 0:
  print('a is zero')
else:
  print('a is negative')

>> a is positive
```

elif and else are not mandatory and can be omitted if not needed.

Input from the keyboard

`input` - Input is a procedure that prints a string to the screen, reads an input and returns the typed text:

```
name = input('Name? ')
print(f'Hi {name}')
>> Name? Johan
Hi Johan
```

Loops and variable mutation

```
i = 10
while i > 0:
  if i % 2 == 0:
    print(i)
  i = i - 1
```

The while-loop continues its execution until the condition (here i > 0) is False.

Indentation (distance to the left margin)

I Python, the indentation of the code carries significant semantic meaning.

```
if a > 0:
  print('a', end='')
  print('b', end='')
```

Prints 'ab', if a > 0, does nothing otherwise

```
If a > 0:
  print('a', end='')
print('b', end='')
```

Prints 'ab', if a > 0, ‘b’ otherwise

Loops within loops

```
i = 0
while i < 3:
  j = 0
  while j < 4:
    print(f'({i}, {j})')
    j = j + 1
  i = i + 1
```

Loops within loops

```
i = 0
while i < 3:
  j = 0
  while j < 4:
    print(f'({i}, {j})')
    j = j + 1
  i = i + 1
```

Output:

```
(0, 0)
(0, 1)
(0, 2)
(0, 3)
(1, 0)
(1, 1)
(1, 2)
(1, 3)
(2, 0)
(2, 1)
(2, 2)
(2, 3)
```

Example: Factorial (floating-point)

```
fac = 1.0
n = 7
while n > 0:
  fac = fac * n
  n = n - 1

print(fac)

>>5040.0
```

Example: Factorial (floating-point)

```
fac = 1.0
n = 1000
while n > 0:
  fac = fac * n
  n = n - 1

print(fac)

>>inf
```

Example: Factorial (integer)

```
fac = 1
n = 1000
while n > 0:
  fac = fac * n
  n = n - 1
print(fac)
```

```
40238726007709377354370243392300398571937486421071463254379991042993851239862902059204420848696940480047998861019719605863166687299480855890132382966994459099742450408707375991882362
77271887325197795059509952761208749754624970436014182780946464962910563938874378864873371191810458257836478499770124766328898359557354325131853239584630755574091142624174743493475534
28646576611667797396668820291207379143853719588249808126867838374559731746136085379534524221586593201928090878297308431392844403281231558611036976801357304216168747609675871348312025
47858932076716913244842623613141250878020800026168315102734182797770478463586817016436502415369139828126481021309276124489635992870511496497541990934222156683257208082133318611681155
36158365469840467089756029009505376164758477284218896796462449451607653534081989013854424879849599533191017233555566021394503997362807501378376153071277619268490343526252000158885351
47331611702103968175921510907788019393178114194545257223865541461062892187960223838971476088506276862967146674697562911234082439208160153780889893964518263243671616762179168909779911
90375403127462228998800519544441428201218736174599264295658174662830295557029902432415318161721046583203678690611726015878352075151628422554026517048330422614397428693306169089796848
25901254583271682264580665267699586526822728070757813918581788896522081643483448259932660433676601769996128318607883861502794659551311565520360939881806121385586003014356945272242063
44631797460594682573103790084024432438465657245014402821885252470935190620929023136493273497565513958720559654228749774011413346962715422845862377387538230483865688976461927383814900
14076731044664025989949022222176590433990188601856652648506179970235619389701786004081188972991831102117122984590164192106888438712185564612496079872290851929681937238864261483965738
22911231250241866493531439701374285319266498753372189406942814341185201580141233448280150513996942901534830776445690990731524332782882698646027898643211390835062170950025973898635542
77196742822248757586765752344220207573630569498825087968928162753848863396909959826280956121450994871701244516461260379029309120889086942028510640182154399457156805941872748998094254
7421735824010636774045957417851608292301353580818400969963725242305608559037006242712434169090041536901059339838357779394109700277534720000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000
```

Listor

Often you want to/require access to more than a limited number of values, easily accessible without requiring a name for each. Then a list is the most common data-structure.

```
a = [1, 1, 2, 3, 5, 8, 13, 21]
print(a)
print(type(a))

>> [1, 1, 2, 3, 5, 8, 13, 21]
<class = 'list'>
```

Listor

```
a = [1, 1, 2, 3, 5, 8, 13, 21]
i = 0
while i < len(a):
  x = a[i]
  print(x**2)
  i = i + 1
```

Listor

```
a = [1, 1, 2, 3, 5, 8, 13, 21]
i = 0
while i < len(a):
  x = a[i]
  print(x**2)
  i = i + 1
```

Output

```
>> 1
1
4
9
25
64
169
441
```

Listor

```
a = [1, 1, 2, 3, 5, 8, 13, 21]
b = []
i = 0
while i < len(a):
  x = a[i]
  b.append(x**2)
  i = i + 1

print(b)
```

Resultat

```
>> [1, 1, 4, 9, 25, 64, 169,
441]
```

Iteration over lists, in reverse

```
a = [1, 1, 2, 3, 5, 8, 13, 21]
i = len(a)-1
while i >= 0:
    print(a[i])
    i = i - 1
```

Output

```
>> 21
13
8
5
3
2
1
1
```

Lists

len(list) - gives the length of the list, in number of items
list[0] - gives the first element in the list
list[len(lista)-1] or list[-1] - gives the last element in the list

Python uses 0-based indexing, unlike e.g. Matlab that uses 1-based indexing.

for-loop

A significant problem with the while-loop is that it invites logical errors.
See the following code:

```
i = 0
while i < 10:
  print(i, end='')
```

What happens when we run it?

for-loop

A significant problem with the while-loop is that it invites logical errors.
See the following code:

```
i = 0
while i < 10:
  print(i, end='')
```

What happens when we run it?

```
>> 000000000000000000000000000000000000000000000000000…
```

for-loop

A significant problem with the while-loop is that it invites logical errors.
See the following code:

```
i = 0
while i < 10:
  print(i, end='')
```

The problem is that iterations with the while-loop often requires three different pieces of code to be correct:
Initialization of loop variable (i)
The loop condition
Updating of the loop variable (i)

for-loop (a better loop, most of the time)

```
for i in range(10):
    print(i, end='')
```

Resulterar i:

```
>> 0123456789
```

range(10) - creates a sequence of integers from 0 up to (but not including) 10

A half-open interval [0, 10).

for-loop

You can iterate directly over lists:

```
a = [1, 3, 7, 11, 13]

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

```
Results in:
>>
1
3
7
11
13
```

Here you don't need to think about indexing at all. The for-loop takes care of this for you.

for-loop with range (in reverse)

It is also possible to create ranges which count in reverse:

```
for i in range(9, -1, -1):
    print(i, end=' ')

>> 9 8 7 6 5 4 3 2 1 0
```

for-loop with range (in reverse)

It is also possible to create ranges which count in reverse:

Start value

```
for i in range(9, -1, -1):
    print(i, end=' ')
```

Step

Stop value

```
>> 9 8 7 6 5 4 3 2 1 0
```

Data-types

Data-types can be thought of as sets of values that a variable can take on, and the basic-operations that can be applied to those values.

Today we have seen several different data-types:
- Integers (int)
- Floating-point numbers / decimal numbers (float)
- Textsequences / strings (string)
- Lists
- Logical, boolean, values (bool) : False / True

Data-types

In many programming languages, one must declare the data-type of a variable, and after that it can not take on values of any other data-type. In Python this is not the case. You can (but it is often poor coding practice to) assign values of different types to a single variable at different occations:

```
a = 7 # Now a is an integer
a = 'hello' # Now a is a string
a = 3.14 # Now a is a float
```

Terminology

Assignment - Change of a variable’s value

Sequence - The order of operations of a program

Iteration - Repetition of a process (e.g. `while` / `for`)

Selection - Choice of operations to perform based on logical conditions