

Lektion 2

Date: 200902

1DV501, Grundläggande programmering

Fredrik Ahlgren

Agenda

- Variabler
- Print statements
- Strings
- Number types
- Type conversions
- Integer operations
- Read text from input

Läsinstruktioner:

- Kap. 2.1-2.8 och 3.1-3.8 Python Fundamentals, Halterman. Övningar:
 - Exercises 5-12 in Assignment 1
-

Variabler, print()

- Variabler sätts med '='
- Python är "intelligent", behöver ej fördefiniera (nedan *integer*)

Example 1:

```
a = 10
b = 5
print(a+b)
```

Example 2:

```
a = 10
b = a + 7
print(a, b)
```

In [1]:

```
# Exempel 1

a = 10
b = 5
print(a+b)
```

15

In [2]:

```
# Exempel 2

10= 10
b = a + 7
print(a, b)
```

10 17

In [5]:

```
d = 1
```

In [8]:

```
print(d)
```

1

- `a, b` --> *variabler*.
 - `a = 10` --> assignment.
 - `b = a + 7` --> `a` används för att assigna `b`
 - Variabler måste **assignas** före de används (ej defineras)
 - Variabel ensam på vänster sida
 - `a + 1 = 10` # ERROR!
-
- Variabler är 'containers' för data
 - En variabel skapas när ett värde 'assignas'.
-

Specifika regler för Python

- Variable names must start with a letter or the underscore character ()
- A variable name cannot start with a number
- A variable name **can only contain alpha-numeric characters and underscores** (a-z, A-Z, 0-9, and)
- Python has certain reserved keywords, which can not be used for variables
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Python naming conventions (Recommended, not a rule):

- Use a lowercase single letter, word, or words.
- Separate words with underscores to improve readability.
- Examples: `x` , `height` , `my_variable`
- Use English, avoid strange characters like the Swedish ÅÄÖ or other special letters

Important

Python keywords can not be used as variable names.

`False, await, else, import, pass, None, break, except, in, raise, True, class, is, return, pass, global, and, for, while, elif, or, with ... etc ..`

Function print()

```
print(100)
print(1, end=" ")
print(2, end="x")
print(3, end="A")
print(4, end="\n")
```

- `print(100)` default, prints 100 and breaks the line
- `print(1, end=" ")` prints 1 with no line break
- `print(3, end="A")` ends with A rather than line break
- `print(4, end="\n")` same as `print(4)`, `\n` symbolizes line break
- `print(10,20,30,40,50)` default, prints with a whitespace between each element and breaks the line
- `print(1,2,3,4,5, sep=", ")` replaces separating whitespace with ", " (a comma plus whitespace)

In [10]:

```
print(100)
print(1, end=" ")
print(2, end="x")
print(3, end="A")
print(4, end="\n")
print(10,20,30,40,50)
print(1,2,3,4,5, sep=", nästa nummer: ")
```

```
100
12x3A4
10 20 30 40 50
1, nästa nummer: 2, nästa nummer: 3, nästa nummer: 4, nästa nummer
: 5
```

Multiple assignments

One assignment on each line (the standard approach):

```
a = 1
b = 2
c = 3
print(a,b,c) # Output: 1 2 3
```

Equivalent using multi-assignment (a more compact version):

```
a,b,c = 1,2,3
print(a,b,c) # Output: 1,2,3
```

- Using multi assignments like `a,b,c = 1,2,3` you can assign multiple variables in a single row.
- Use it only in trivial assignments like the one above, avoid it when having more complex expressions on the right-hand side.

In [12]:

```
a = 1
b = 2
c = 3
print(a,b,c)
```

1 2 3

In [15]:

```
a,b,c = 1,2,3
print(a,b,c)
```

1 2 3

Strings

- A string is a sequence of characters
- character = letter, digit, whitespace, ... (all sorts of symbols)
- String examples:

```
"Are you suggesting that coconuts migrate?"
```

```
'Not at all. They could be carried.'
```

```
"What? A swallow carrying a coconut?"
```

- A string is created using `"..."` or `'...'`

In [19]:

```
s1 = 'Dead Collector: Ere, he says he's not dead.\n' # Using ''
s2 = "Large Man: Yes he is.\n" # Using "..."
s3 = " 'Dead' Man\": I'm not.\n" # Note the Escape-character!
print(s1,s2,s3)
```

```
Dead Collector: Ere, he says he's not dead.
Large Man: Yes he is.
'Dead' Man": I'm not.
```

String Concatenation

In [20]:

```
print("This sentence gives a string that
      according to me is too long to fit
      on a single line.")
```

```
File "<ipython-input-20-b63cffc18e55>", line 1
    print("This sentence gives a string that
          ^
```

SyntaxError: EOL while scanning string literal

- You can **not** break a line in the middle of a string.
- However, you can create a new string by adding two strings this is called string concatenation

In [21]:

```
print("This sentence gives a string that "
      + "according to me is too long to fit "
      + "on a single line.")
```

```
This sentence gives a string that according to me is too long to f
it on a single line.
```

In [25]:

```
print("This sentence gives a string that "+
      "according to me is too long to fit "+
      "on a single line.")
```

```
This sentence gives a string that according to me is too long to f
it on a single line.
```

Notice: string + string = new string

```
s = 'A'
s = s + 'B'
s = s + 'C'
print(s) # Output: ABC
```

In [26]:

```
s = 'A'
s = s + 'B'
s = s + 'C'
print(s) # Output: ABC
```

ABC

Escape Sequences

In [27]:

```
print("Black Knight: "It's just a flesh wound.") # Error!
```

```
File "<ipython-input-27-8eef0903064c>", line 1
    print("Black Knight: "It's just a flesh wound.") # Error!
                          ^
```

SyntaxError: invalid syntax

- It is interpreted as: "And then she said: " followed by something strange) Strings created with "..." can not contain the character "
- Escape Sequences: Characters representing another character
- Most frequently used escape sequences in Python

Escape Represents:

- \" "
- \' '
- \\
- \n line break
- \t tab

In [28]:

```
print("Black Knight: \"It's just a flesh wound.\")
```

Black Knight: "It's just a flesh wound."

or ... by using

In [29]:

```
print('Black Knight: "It´s just a flesh wound."')
```

Black Knight: "It´s just a flesh wound."

Escape examples

In [30]:

```
print("One\n \tTwo\n \t\tThree\n")
```

```
One
    Two
        Three
```

In [31]:

```
print("King Arthur:\n\tLook stop that!:\n"+"Black Knight:\n\tChicken.\n"+"King Arthur:"+"\n\tLook, I'll Have your Leg.\n"+"Black Knight:"+"\n\tRight, I'll do you for that.")
```

```
King Arthur:
    Look stop that!:
Black Knight:
    Chicken.
King Arthur:
    Look, I'll Have your Leg.
Black Knight:
    Right, I'll do you for that.
```

In [32]:

```
# Readability counts!

print("King Arthur:\n\tLook stop that!:\n"+
      "Black Knight:\n\tChicken.\n"+
      "King Arthur:"+
      "\n\tLook, I'll Have your Leg.\n"+
      "Black Knight:"
      +"\n\tRight, I'll do you for that.")
```

```
King Arthur:
    Look stop that!:
Black Knight:
    Chicken.
King Arthur:
    Look, I'll Have your Leg.
Black Knight:
    Right, I'll do you for that.
```

Printing formatted strings

- Output: height 3.5 and width 4.0 gives area 14.0 (in all three cases)

In [35]:

```
# Using multiprint
h = 3.5
w = 4.
print("height",h,"and width",w,"gives area",h*w)

# Using format
print("height {0} and width {1} gives area {2}".format(h,w,h*w))

# Using f-strings
print(f"height {h} and width {w} gives area {h*w}")
```

```
height 3.5 and width 4.0 gives area 14.0
height 3.5 and width 4.0 gives area 14.0
height 3.5 and width 4.0 gives area 14.0
```

- The 2nd approach (formatted print) is now outdated and replaced with the 3rd approach
- However, the textbook by Halterman sometimes uses the formatted print approach) you should understand it. Introduced in section 2.8.
- **We recommend 1st and 3rd approach**

Variables and Types

In [36]:

```
s = 'Peasant: Oh, she turned me into a newt!'
print(s, type(s))
```

Peasant: Oh, she turned me into a newt! <class 'str'>

In [39]:

```
s = 1.
type(s)
```

Out[39]:

float

In [40]:

```
s = 42
print(s, type(s))
```

42 <class 'int'>

In [41]:

```
s = 42. # note the decimal
print(s, type(s))
```

42.0 <class 'float'>

- A variable has a current type depending on what type of value it contains
- You can question a variable `s` of its current type using `type(s)`
- The `type` function can also be applied on values

In [42]:

```
# Simple math?
print(3.3 - 1.1)

# The answer .. ?
```

2.1999999999999997

Mixing types

In [43]:

```
text = 'Peasant: And the hat. ' + 'She´s a witch!!!' # OK!
n = 7 + 8 # OK!
```

In [44]:

```
result = "Height = " + 123 # Error!
```

```
-----
TypeError                                Traceback (most recent c
all last)
```

```
<ipython-input-44-7020183896d2> in <module>
----> 1 result = "Height = " + 123 # Error!
```

TypeError: can only concatenate str (not "int") to str

- You can not concatenate a string with an integer
- In general, you can not mix types in expressions
- However, you can convert an integer to string and then concatenate them

In [45]:

```
h = 123
result = "Height = " + str(h) # Converts h to a string
print(result, type(result) )
```

```
Height = 123 <class 'str'>
```

Type Conversion

- `str(n)` converts a number `n` to a string
- `int(s)` converts a string `s` to an integer

Example

In [48]:

```
s = str(123) # Convert integer to string
print(s, type(s))
n = int("123") # Convert string to integer
print(n, type(n))
```

```
123 <class 'str'>
123 <class 'int'>
```

- You will get an error if you try to convert an arbitrary string (e.g. "hello") to an integer.

Decimal Numbers (1)

- Python can of course also handle decimal numbers

In [53]:

```
pi = 3.14159265359
print(pi, type(pi))
d = 2*3
print(d, type(d))
Na = 6.02214076e23 # Avogadros number
print(Na, type(Na))
```

```
3.14159265359 <class 'float'>
6 <class 'int'>
6.02214076e+23 <class 'float'>
```

- Decimal numbers are called float in Python
- 6.02214076e23 should be interpreted as $6.02214076 \cdot 10^{23}$

Decimal Numbers (2)

In [54]:

```
print( 4/2 , type(4/2))
print( 4*2.0 , type(4*2.0))
print( 4+2.0 , type(4+2.0))
```

```
2.0 <class 'float'>
8.0 <class 'float'>
6.0 <class 'float'>
```

- Operations involving floats always give a float
- Division using `=` always give a float

Decimal Numbers (3)

- The function `round()` is used to round of decimals

In [55]:

```
d = 1234.56789
x = int(d) # Convert to integer
print(x, type(x))
x = round(d) # Correctly rounded off
print(x, type(x))
x = round(d,2) # Two decimals
print(x, type(x))
```

```
1234 <class 'int'>
1235 <class 'int'>
1234.57 <class 'float'>
```

- Convert to integer using `int(...)` cuts off decimals
- `round(d)` correctly rounded off integer
- `round(d,2)` two decimal float correctly rounded

Integer Division and Modulus

- Integer Division `A // B` -> how many B fit inside A?

Integer division Example:

In [56]:

```
7 // 3
```

Out[56]:

```
2
```

In [57]:

```
27 // 4
```

Out[57]:

```
6
```

In [58]:

```
20 // 4
```

Out[58]:

```
5
```

In [59]:

```
9 // 10
```

Out[59]:

0

In [60]:

```
94 // 10
```

Out[60]:

9

In [61]:

```
(-94) // 10
```

Out[61]:

-10

- Wait ... what happened here ???

<http://python-history.blogspot.com/2010/08/why-pythons-integer-division-floors.html> (<http://python-history.blogspot.com/2010/08/why-pythons-integer-division-floors.html>)

But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity)

Modulus

- remainder of A in A/B

In [62]:

```
13%3
```

Out[62]:

1

In [63]:

```
15%5
```

Out[63]:

0

In [64]:

```
16%5
```

Out[64]:

```
1
```

In [65]:

```
17%5
```

Out[65]:

```
2
```

In [66]:

```
77%10
```

Out[66]:

```
7
```

Example - Arithmetics

In [67]:

```
x = 2.5
y = x + 50.0
z = x * y

print("X =", x, ", Y =", y, ", Z =", z)

m = 17
n = 5

div = m // n

mod = m % n

print("\nDivide:", div, ", Modulus:", mod)
```

```
X = 2.5 , Y = 52.5 , Z = 131.25
```

```
Divide: 3 , Modulus: 2
```

Operator shortcuts

- Python allows certain short cuts for frequently used statements.
- The following statements

In [68]:

```
n = n + 1 # Increase value of n by 1
m = m - 1 # Decrease value of m by 1
a = a + 2 # Increase value of a by 2
```

- can also be written as

In []:

```
n += 1 # Increase value of n by 1
m -= 1 # Decrease value of m by 1
a += 2 # Increase value of a by 2
```

- The short cuts above also works for other operators like *,/,%, ...
- Many other programming languages use n++ (and) to increase/decrease by 1, this does not work in Python.

Reading User Input

In [71]:

```
s = input('Guard: Halt! Who goes there? ')
print(s)
print('Type: ', type(s))
```

```
King Arthur
Type:  <class 'str'>
```

- We can read user input using the function input()
- When executed, it halts execution and waits for user input (from the keyboard)
- input() always returns a string

Reading Numbers

In [72]:

```
# Read string and then convert to integer
s = input("Enter an integer: ") # A string like "123"
n = int(s) # Convert to int
# Read and convert in one line
m = int( input("Enter another integer: ") ) # Shortcut
print(n, " + ", m, " = ", n+m)
```

12 + 34 = 46

- `input()` returns a string -> must be converted before we continue
- `int( input(...) )` -> read and convert in one go

Programming Examples - BMI

Exercise

Write a program `bmi.py` which computes the BMI (Body Mass Index) for a person. The program will read length and weight from the keyboard and then present the result as output. The BMI is computed as $\text{weight}/(\text{length})^2$, where the length is given in meters and the weight in kilograms. The BMI is always a (correctly rounded) integer.

- Weight and length are floats, BMI is an int
- How to round of a float to an integer?
- Try to have a plan before you start to program
- Take small steps) add a few lines ...
- ... and print intermediate results along the way

In [73]:

```
# Read input
length = float( input("Your length in meters: ") )
weight = float( input("Your weight in kilograms: ") )
# Compute BMI
bmi = weight/length**2
bmi = round(bmi)
# Present result
print("\nYour BMI is",bmi)
```

Your BMI is 23

Programming Examples - Pick a digit

Exercise

Write a program `pickdigit.py` which reads a three-digit integer and then prints the middle digit.

An example of an execution:

```
Enter a three-digit integer: 257
The second digit is 5
```

- Q: How to extract middle digit from an integer?

...

- A: Remove last digit with integer division ($n // 10$) and
- ... then pick the last remaining digit with modulus ($n \% 10$)

Pick a digit - Solution

In [74]:

```
# Read input, an integer
n = int( input("Enter a three-digit number: "))
# Pick mid digit from n = 234
n = n//10 # n = 23
n = n%10 # n = 3
# Present result
print("The second digit is",n)
```

The second digit is 4

Note

- Try to have a plan before you start to program
- Attack hard parts/problems first,
- then add the remaining parts

About the assignment exercises ...

We have two goals with our assignment exercises

1. Practice Python programming

- a lot of exercises using different language constructs

1. Enhance problem solving skills

- Exercises often comes with a problem to solve

Suggested solution strategy

1. Solve the problem -> a solution idea (maybe a sketch on the paper)
 2. Translate your solution to a Python program
- Do not start programming without a solution idea, it will most likely not show up as a miracle while programming
 - Solve the hard part first, add "read input" and "present result" afterwards

Until next lecture ...

- Start working on the Lecture 2 exercises in Assignment 1
- Visit the tutoring sessions
- Bring your laptop to the tutoring sessions

For each lecture (recommended)

- Read corresponding sections in textbook by Halterman
- Read and understand the lecture slides
- Work on the corresponding programming exercises

In []: