Getting Started

1DV501/1DT901: Introduction to programming

Jonas Lundberg, office B3024

Jonas.Lundberg@lnu.se

The slides are available in Moodle

August 30, 2022

But first ... prepare VS Code for Assignment 1

Suggested folder structure

```
jlnmsi (my home folder)
    software
        python_courses
            1DV501 (or 1DT901)
```

Create this structure using your operating system. We suggest that you place the folder `software` close to (or inside) your root folder. Then do the following:

1. Start VS Code
2. Use File → Open... and select folder `1DV501` ⇒ `1DV501` shows up in VC Code
3. Right-click on `1DV501` and select New Folder and add `your_lnu_username_assign1` as a new folder inside `1DV501`
4. Right-click on `your_lnu_username_assign1` and select New File and add `hello.py` as a new file ⇒ the file `hello.py` opens up in the editor
5. Enter program code (e.g. `print("Hello")` in the editor window
6. Right-click on the editor window and select: `Run Python File in Terminal`. The result, if everything works, is that `Hello` is printed on your terminal output.
7. Repeat 4. for each new exercise
8. Repeat 3. for each new assignment

Today ...

Aim: Write small programs. Learn about strings, arithmetics, printing results on the screen, and reading input from the keyboard.

Content

- Variables
- Print statements
- Strings
- Number types
- Type Conversions
- Integer operators
- Read text from keyboard

Reading Instructions: Sections 2.1-2.8 and 3.1-3.8 in the textbook by Richard L. Halterman
Exercises: Lecture 2 exercises in Assignment 1

Example: Variables and `print()`

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

Output: 15

```
# Example
a = 10
b = a + 7
print(a, b)    # Multi-print
```

Output: 10 17

Notice

- The code fragments shown above are actually complete Python programs
- You can create a Python file (e.g. `temp.py`), insert the code fragments, and execute them
- This holds for all the code fragments we will show in this lecture. They are complete executable Python programs

Variables (1)

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

Output: 15

```
# Example
a = 10
b = a + 7
print(a, b)    # Multi-print
```

Output: 10 17

- `a, b` are **variables** ⇒ named containers for storing data values
- Vocabulary
  - We refer to `a = 10` as an **assignment**
  - `a = 10` ⇒ we **assign** variable a the value 10
  - `b = a + 7` ⇒ we **use** variable a to assign variable b a new value
- Variables must be assigned a value before they are used
- Variable must be alone on the left-hand side

```
a + 1 = 10    # Error!!!
```

- Variables can be reused ⇒ assigned new values multiple times

Variables (2)

- Rules for Python variables (Error if not following them)
  - A variable name can only contain alpha-numeric characters and underscores (a-z, A-Z, 0-9, and _)
  - Variable names must start with a letter or the underscore character (_)
  - A variable name cannot start with a number
  - 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 åäöÅÄÖ

Variables (3)

Python keywords ⇒ can not be used as variable names ⇒ the compiler will give an error if you try

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

About half of the keywords above will be presented and used in this course

The function print()

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

Output
100
12x3A4

```
# Multiprint
print(10, 20, 30, 40, 50)
print(1, 2, 3, 4, 5, sep=", ")
```

Output
10 20 30 40 50
1, 2, 3, 4, 5

- `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)

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.

Strings

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

```
"Hello World!"      "Happy days are here again!"      "x"

"(GD&D   .,~-{]HG()B( SG-,.M*PNI \\\n\t\n"
```

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

```
s1 = 'Python is fun!'       # Using '...'
s2 = "Isn't it?"            # Using "..."
print(s1, s2)
```

Output
Python is fun! Isn't it?

String Concatenation

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

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

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

- Notice: `string + string = new string`

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

String Multiplication

```
s = "Hello"
print(s)            # Output: Hello

s = 3 * "Hello"
print(s)            # Output: HelloHelloHello
```

- You can multiply a string with an integer.
- `3 * "Hello"` $\Rightarrow$ concatenate `Hello` three times with itself
- This is called **string multiplication**

Escape Sequences

```
print("And then she said: "Hello Darling".") # Error!
```

- 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

- That is, we should have written

```
print("And then she said: \"Hello Darling\".")   # OK!
```

- Or by using an enclosing `'...'`

```
print('And then she said: "Hello Darling".')  # Also OK!
```

Escape Example

```
# Using tab (\t) and linebreak (\n)

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

print("CS Teachers:\n"+"\tJesper\n"+"\tJonas\n"+"\tOla")
```

Output:

```
One
    Two
        Three

CS Teachers:
    Jesper
    Jonas
    Ola
```

Printing formatted strings

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

```
# Using multiprint
h = 3.5      # Rectangle dimensions
w = 4.0
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}")
```

- The 2nd approach (formatted print) is now outdated and replaced with the 3rd approach (f-strings)
- 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

```
s = 'Hello!'
print(s, type(s))

s = 7
print(s, type(s))
```

Output:

```
Hello! <class 'str'>
7 <class 'int'>
```

- 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

```
print(type("Hello"), type(7))
```

Output: `<class 'str'> <class 'int'>`

Mixing Types

```
text = 'Hello ' + 'World!'        # OK!
n = 7 + 8                         # OK!
result = "Height = " + 123        # Error!
print(text, n, result)
```

Output:

```
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

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

Output: `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

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

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

Output:
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

```
pi = 3.14159265359
print(pi, type(pi))

d = 2/3
print(d, type(d))

Na = 6.02214076e23      # Avogadro's number
print(Na, type(Na))
```

Output:

```
3.14159265359 <class 'float'>
0.6666666666666666 <class 'float'>
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)

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

Output:

```
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 built-in function `round()` is used to round off decimals

```
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))
```

Output:

```
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 off

Programming Example: Circle Area

The area A of a circle with radius r can be computed as

$$A = \pi \cdot r^2 \quad \text{where} \quad \pi \approx 3.141593.$$

Compute the area of a circle with radius $r = 5.3$ and present the result with 3 decimals (correctly rounded off).

Solution: Circle Area

```
pi = 3.141593
r = 5.3

A = pi*r*r              # Compute area
#A = pi*r**2            # Using ** (raised to)
res = round(A,3)        # Round off

print("Circle area:", res)
```

Output: Circle area: 88.247

A 10 minute break?

zzzzzzzzzzzzz ...

Integer Division and Modulus

Integer Division $A//B \Rightarrow$ how many B fit inside A?

- Example

$$7//3 = 2, \quad 27//4 = 6, \quad 20//4 = 5, \quad 9//10 = 0, \quad -94//10 = -9$$

- **Exercise**: Compute a) $13//3$, b) $17//5$, c) $4//5$, d) $16/4$
- Answer: a) 4, b) 3, c) 0, d) 4

Modulus $A\%B \Rightarrow$ left-overs of A in $A//B$. Formally

$$A\%B = A - \frac{A}{B} \cdot B$$

- Example

$$15\%4 = 15 - \frac{15}{4} \cdot 4 = 15 - 3 \cdot 4 = 3$$

- More examples

$$7\%3 = 1, \quad 27\%4 = 3, \quad 20\%4 = 0, \quad 5\%10 = 5$$

- **Exercise**: Compute a) $13\%3$, b) $15\%5$, c) $4\%5$, d) $77\%10$
- Answer: a) 1, b) 0, c) 4, d) 7

Hint: think in terms of distances.

Example - Arithmetics

```
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)
```

Output:

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

Divide: 3 , Modulus: 2
```

Operator short cuts

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

```
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

```
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 `n--`) to increase/decrease by 1. This does not work in Python.

Reading User Input

```
s = input('Please enter some text: ')
print('Entered data: ',s)
print('Type: ', type(s))
```

Usage:

```
Please enter some text: Jonas Lundberg
Entered data: Jonas Lundberg
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
- The provided text (`'Please enter some text: '`) becomes a user instruction

Reading Numbers

```
# Read string and then convert to integer
s = input("Enter an integer: ")    # A string like "123"
n = int(s)                         # Convert "123" to int 123

# Read and convert in one line
m = int(input("Enter another integer: "))  # Shortcut

print(n, " + ", m, " = ", n+m)
```

Usage:

```
Enter an integer: 23
Enter another integer: 64
23  +  64  =  87
```

- `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 $weight/(length)^2$, where the length is given in meters and the weight in kilograms. The BMI is always a (correctly rounded off) integer. An example of an execution:

```
Your length in meters: 1.83
Your weight in kilograms: 83

Your BMI is: 25
```

Notice

- Weight and length are `floats`, BMI is an `int`
- How to round off a float to an integer?

BMI - Solution

```
# 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)
```

Usage:

```
Your length in meters: 1.83
Your weight in kilograms: 88

Your BMI is 26
```

- Try to have a plan before you start to program
- Take small steps ⇒ add a few lines ...
- ... and print intermediate results along the way

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
```

Question

- 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

```
# 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)
```

Usage:

```
Enter a three-digit number: 567
The second digit is 6
```

- 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
   $\Rightarrow$ a lot of exercises using different language constructs
2. Enhance problem solving skills
   $\Rightarrow$ Exercises often comes with a problem to solve

Suggested solution strategy

1. Solve the problem $\Rightarrow$ a solution idea
   $\Rightarrow$ 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
- Visit Python Help Desk (campus Växjö and online) if you have problems with your Python and VS Code installations

For each lecture (recommended)

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