

Introduction to Python Programming

(3) Variables, Types, Operators

S. Thater and A. Friedrich

Saarland University

Winter Semester 2011/2012

- What is an **algorithm**?
- What is a **program**?
- Requirements for algorithms?
- What is **compilation**?
- What is **interpretation**?
- What does **platform independence** mean?

Imperative Programming Paradigm

- **Imperative:** *First do this, then do this.*

Procedural Programming. Control Structures execute computational steps, state of the program changes as a function of time.

Commands can be grouped into procedures.

Example

```
Celsius_to_Fahrenheit (c)
```

- 1 Multiply `c` with 1.8 and save result to `temp`.
- 2 Add 32 to `temp` and return result of this.

Elements of imperative programs

- Variables
- Assignments
- Expressions
- Control Structures: loops, branches

Values, Variables, Data Types

- **Values** may have different **data types**: numbers, lists, strings...

Variable Assignment

```
list = [1, 2, 3, 4]
```

```
number = 4
```

```
text = 'hello'
```

```
number = 'world'
```

number

text

The diagram illustrates memory storage. A 2x6 grid of cells is shown. The top row contains the values 4, 8, and three empty cells. The bottom row contains the characters 'h', 'e', 'l', 'l', 'o', and '\0'. An arrow labeled 'number' points to the first cell of the top row (containing '4'). An arrow labeled 'text' points to the first cell of the bottom row (containing 'h').

	4		8		
h	e	l	l	o	\0

- **Variables** = **placeholders** for values.
- **Variables** point to positions in the memory where values are stored. Value of a variable can change over time.

Some Data Types

- **Boolean:** truth values: `True` and `False`
- **Numbers:** `int` (`2`), `float` (`2.0`), `complex`
- **Strings:** `str`
- **Collections:** `tuple`, `list`, `set`, `dict`

Dynamic Typing

Variables in Python do not have fixed data types.

- The type of a variable is the assigned value's data type.
- During runtime, a variable can take values of different types.

Dynamic Typing

```
>>> x = 15.4
>>> type(x)
<type 'float'>
>>> x = "Python is great!"
>>> type(x)
<type 'str'>
```

Floating Point Numbers

- Decimal numbers are represented as **floats**
(1.1, 47.11)
- Range depends on system
- **CAREFUL!** Often, the internal representation of floating point numbers is imprecise.

```
>>> 0.1  
0.1000000000000000001
```

- What to do about this? \Rightarrow use ϵ when comparing floating point numbers.

```
>>> epsilon = 0.000000000000000001  
>>> x_equal_y = abs(x-y) < epsilon  
>>> x_equal_y  
True
```


Expressions

- **Expressions** = constructs describing a value
- We distinguish:
 - ▶ **Literals** = expressions from / in which the value can be directly read / written, e.g. `1.0`, `True`, `"Hello World"`
 - ▶ **Variables** = references to values
 - ▶ Complex expressions with operators, e.g. `3+5`
 - ▶ Calls of functions or methods, e.g.
`find_max(L1)`

Elementary Arithmetic Operators

Addition	$a + b$
Subtraction	$a - b$
Multiplication	$a * b$
Division	a / b
Modulo	$a \% b$

- If a and b do not have the same type, the operations result in a value of the more general type.

Example

```
>>> a = 1
>>> b = 2.4
>>> a + b
3.3999999999999999
```

What are the types in this example?
Which type is more general?
Why?

Precedence

- Expressions may contain multiple operators: $3 + 2 * 4$
- **Precedence** = order in which operators are evaluated
- Standard precedence rules: multiplication/division before addition/subtraction
- Parentheses indicate precedence directly

Example

$$3 + (2 * 4) = 11$$

$$(3 + 2) * 4 = 20$$

- Style: sometimes it is recommended to use parentheses even if they are redundant (legibility)
- Don't use parentheses when precedence is irrelevant, e.g. $2 + 3 + 4$ is better than $2 + (3 + 4)$

Truth Values (Boolean)

- The type `bool` represents the two truth values `True` and `False`

negation	<code>not a</code>
conjunction	<code>a and b</code>
disjunction	<code>a or b</code>

- On board: truth tables
- Precedence: `not` > `and` > `or`
`a and not b or c = (a and (not b)) or c`
- Short-circuit evaluation**: the evaluation stops as soon as the result is evident (`True or ...`)

String Literals

- Strings are sequences of characters (no separate type for characters).

Example

```
"This is a string."  
'That, too.'  
"He said \"Hello.\"."  
'He said "Hello".'
```

- String may not contain any special characters (umlauts etc.) if no encoding is specified. (in Python 2.x - no problems in Python 3.x!)
- Encoding is specified in the first code line:

```
# -*- Coding: utf-8 -*-  
# -*- Coding: latin-1 -*-
```

Lists

- We can have lists of numbers: `numberList = [1, 2, 3, 4]`
- Or lists of strings:
`weekdays = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']`
- We can access individual items of the list using brackets:

```
1 >>> print(weekdays[2])
2 Wed
3 >>> print(weekdays[6])
4 Sun
```

Index	0	1	2	3	4	5	6
List	'Mon'	'Tue'	'Wed'	'Thu'	'Fri'	'Sat'	'Sun'

Exercise 1: Types

Name the types of the following values.

- (a) `1.0`
- (b) `"a"`
- (c) `False`
- (d) `5`
- (e) `['hello', 'world']`
- (f) `'c'`
- (g) `"Python:"`
- (h) `[2.5, 6.7, 1.2, 4]`

Some String Operators

- Concatenation:

```
'Hello' + 'World' ==> 'HelloWorld'
```

- Access to individual characters with list indices:

```
'Hello'[0] ==> 'H'
```

```
'Hello'[1] ==> 'e'
```

- Test whether a substring occurs:

```
'He' in 'Hello' ==> True
```

```
'Ha' in 'Hello' ==> False
```

- Length: `len('Hello') = 5`

Relational Operators

less than	$a < b$
greater than	$a > b$
less than or equal to	$a \leq b$
greater than or equal to	$a \geq b$
Equal to	$a == b$
not equal to	$a != b$

- The result of such a comparison is a boolean.

Example

```
>>> 3 > 2
```

```
True
```

```
>>> (2*3) + 4 != 2*3 + 4
```

```
False
```

Variables

- Placeholders for values
- one can assign the value of an expression to variables
- variables can be evaluated in order to use their value in an expression
- `print()` is a function that prints the value of an expression to the console (the standard output)

Example

```
>>> number = 123
>>> number = number + 2
>>> print(number)
125
```

Variables

- Variables (more generally, all identifiers) must start with a letter or “_”. The remainder may include digits.
- umlauts etc. are not allowed (ASCII encoding)
(actually allowed in Python 3.0 - but better stick to ASCII anyways)
- the name must not be a keyword (`if`, `while` etc)
- the names are case-sensitive
- convention: variables start with a lower-case letter

Which ones are allowed/recommended?

`foo` , `2foo` , `foo12` , `_foo` , **`if`** , `überzweg`

Assignments I

- `var = expr`
the expression `expr` is evaluated, then its value is stored in `var`.
- `var1 = var2 = ... = expr`
the value of `expr` is assigned to all variables `vari` (all variables point to the same value in the memory)

Example

```
>>> a = b = 6.0/4.0
>>> print(a)
1.5
>>> print(b)
1.5
```

Assignments II

- $\text{var}_1, \text{var}_2, \dots, \text{var}_n = \text{expr}_1, \text{expr}_2, \dots, \text{expr}_n$
all expr_i are evaluated, then the corresponding values are assigned to var_i

Example

```
>>> a, b = 6.0/4.0, 'Hello' + 'World'
>>> print(a)
1.5
>>> print(b)
HelloWorld
```

Assignments

Long version	Shorthand
<code>x = x + expr</code>	<code>x += expr</code>
<code>x = x - expr</code>	<code>x -= expr</code>
<code>x = x * expr</code>	<code>x *= expr</code>
<code>x = x / expr</code>	<code>x /= expr</code>
<code>x = x % expr</code>	<code>x %= expr</code>

Exercise 2: Assignments

The following listing shows several steps of a program. For each step, write down the values and types of `a`, `b` and `c`.

- (a) `a, b = 5, 3`
- (b) `c = 'test'`
- (c) `a = a % b`
- (d) `b *= b`
- (e) `a = c[1]`
- (f) `c = a = b = True`
- (g) `c = not (a or b)`
- (h) `a = b or c`
- (i) `b = str(a) + " Love"`

Type Conversion in Python

Reading a String from the Console

```
variable = input(prompt)
```

- Anything we read from the console is a **string**, even if we enter '42.0'
- For computation, we need to convert this to some **number**.

```
1 variable = input("Enter a number: ")
2 int_number = int(variable)
3 float_number = float(variable)
```

- Conversion of numbers (or other types) to strings works like this:
`text = str(number)`

Exercise 3: Car Stopping Distance

Write a program `car_stopping_dist.py` that computes the stopping distance of a car using the following rule of thumb. (The only input to your program is the velocity, which is to be read from the console.)

- Breaking distance = $(\text{velocity} / 10) * (\text{velocity} / 10)$
- Reaction distance = $(\text{velocity} / 10) * 3$
- Stopping distance = reaction distance + breaking distance

Check whether your program is correct using the following numbers:

Velocity	Stopping Distance
80.0kmh/h	88.0m
50.0kmh/h	40.0m
100.0kmh/h	130.0m

Exercise 4: Volume of a Cone

Write a program `cone_volume.py` that computes the surface area and the volume of a cone.

Radius and height are to be read from the console.

$$\text{SurfaceArea} = \pi * r * h + \pi * r^2$$

$$\text{Volume} = \frac{1}{3} * \pi * r^2 h$$

Radius	Height	Surface Area	Volume
2.0	5.0	43.982297	20.941905
2.7	12.5	128.930962	95.330673

HINT: You can use the following lines of code when using π :

```
1 import math
2 print(math.pi)
```

Exercise 5: Celsius to Fahrenheit Converter

Celsius to Fahrenheit

$$\text{Fahrenheit} = \text{Celsius} * 1.8 + 32$$

- Write a program `celsius_fahrenheit.py` that reads a degree Celsius and outputs the corresponding value in degrees Fahrenheit.
- The Celsius value is to be read from the command line.

Exercise 6: Indian Takeaway

1	Achari Paneer
2	Gajar Ka Achar
3	Aloo Dum
4	Kabuli Chana
5	Baingan Bharta
6	Apple Jalebi

- In an Indian restaurant, the menu items are labeled with numbers, which the customers use to order their dishes. Your job is to translate these numbers to the names of the dish for the cooks.
- Write a program `indian_takeaway.py` that performs this task.

Sample Output

```
>>>
INDIAN TAKEWAY!
Please enter the number of your dish:
4
Thank you for ordering Kabuli Chana
```

Exercise 7:

What is the output of the following program?
Explain what happens here!

```
1 x = (5 == 6)
2 y = str(x)
3 print(y[1])
```