Python for Language Processing

(2a) Variables, Types, Operators

Dr. Jakob Prange

Fakultät für Angewandte Informatik - Universität Augsburg

CL Fall School 24

Credit: This course is based on material developed by
Annemarie Friedrich, Stefan Thater, Michaela Regneri, and Marc Schulder at Saarland University

Recap

- 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

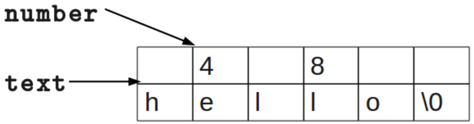
- Variables
- Assignments
- Expressions
- Control Structures: loops, branches

Values, Variables, Data Types

Variable Assignment

```
myList = [1, 2, 3, 4]
number = 4
text = 'hello'
number = 'world'
```

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



- Variables = **placeholders** for values.
- Variables point to positions in the memory where values are stored.
  Value of a variable can change over time. (Point to a different location or overwrite the memory location's value.)

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)
<class 'float'>
>>> x = "Python is great!"
>>> type(x)
<class '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
0.10000000000000001
```

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

```
>>> epsilon = 0.0000000000000001
>>> 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
```

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

Truth Values (Boolean)

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

negation	**not** a
conjunction	a **and** b
disjunction	a **or** b

- Homework: refresh your knowledge on 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.\"."
"""This is a multiline string.
It does not stop at a line break."""
'''Another multiline string.
   Beware of spaces at the line start.'''
```

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:

```
>>> print(weekdays[2])
Wed
>>> print(weekdays[6])
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]`
- (i) `"7"`

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 <= b`
greater than or equal to	`a >= 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 I

- 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 II

- Variables (more generally, all identifiers) must start with a letter or “_”. The remainder may include digits.
- umlauts etc. are allowed in Python 3.X, but we recommend to 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 (see PEP 8 - Style Guide for Pyton Code)

Which ones are allowed/recommended?

```
foo, 2foo, foo12, _foo, if, überzwerg
```

Assignments I

1. `var = expr`
   the expression `expr` is evaluated, then its value is stored in `var`.
2. $\texttt{var}_1$ `=` $\texttt{var}_2$ `= ... = expr`
   the value of `expr` is assigned to all variables $\texttt{var}_i$ (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

3. $\texttt{var}_1$, $\texttt{var}_2$, ..., $\texttt{var}_n$ = $\texttt{expr}_1$, $\texttt{expr}_2$, ..., $\texttt{expr}_n$
   all $\texttt{expr}_i$ are evaluated, then the corresponding values are assigned to $\texttt{var}_i$

Example

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

In general, use option (1). Option (2) and (3) are sometimes used in more complex cases (later in the course!).

Assignments

Long version	Shorthand
`x = x + expr`	`x += expr`
`x = x - expr`	`x -= expr`
`x = x * expr`	`x *= expr`
`x = x / expr`	`x /= expr`
`x = x % expr`	`x %= expr`

Example

```
>>> a = 5
>>> a += 3
>>> print(a)
8
>>> a /= 2
>>> print(a)
4
```

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"`

Example: read text from console

console_input.py

```python
first = input("Input your first name: ")
last = input("Input your last name: ")
middle = input("Input your middle name if you
    have one, or hit enter if you don't have one. ")
print("Your full name is: ", first, middle, last)
```

`variable = input(prompt)`

1. prints the text given in `prompt` on the console
2. waits for the user to enter some text string (terminated by ’enter’)
3. assigns the text string that the user has entered to `variable`

Type Conversion in Python

Reading a String from the Console

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

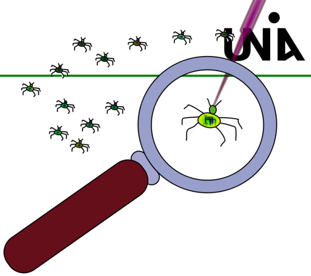
- Anything we read from the console is a **string**, even if we enter '42.0'
- For computation, we need to convert this to some **number**.

```python
variable = input("Enter a number: ")
int_number = int(variable)
float_number = float(variable)
```

- Conversion of numbers (or other types) to strings works like this:

  ```
  text = str(number)
  ```

How to debug (a simple way to start with)



- Don't write down the entire program at once.
- Test after each line:
  - Are there any syntax errors? (IDLE shows them immediately)
    ⇒ Can you run the program?
  - Print out the value of the important variables and check whether they are what you expect!
  - After testing the line, comment out the print statement.

```
variable = input("Enter a number: ")
# print(variable)
int_number = int(variable)
# print(int_number)
float_number = float(variable)
print(float_number)
```

Exercise 3:

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

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

Exercise 4: 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.)

- Braking distance = (velocity / 10) * (velocity / 10)
- Reaction distance = (velocity / 10) * 3
- Stopping distance = reaction distance + braking 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 5: 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.

$$SurfaceArea = \pi * r * h + \pi * r^2$$

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

Radius	Height	Surface Area	Volume
2.0	5.0	43.982297	20.9439510
2.7	12.5	128.930962	95.4258768

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

```python
import math
print(math.pi)
```

Exercise 6: Celsius to Fahrenheit Converter

Celsius to Fahrenheit

Fahrenheit = 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 console.

Exercise 7: 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
```