MI3103
Praktikum Antar Muka Komputer
Pengantar Pemrograman Python

Fadjar Fathurrahman

2018

1 Tujuan

- Dapat membuat dan mengeksekusi program Python sederhana

2 Perangkat lunak yang diperlukan

- Linux OS
- Distribusi Anaconda untuk Python 3
- Browser
- Editor teks seperti gedit, VSCode, Atom

3 Pendahuluan

`http://www.scipy-lectures.org/language/first_steps.html`

4 Mengenal interpreter Python (konsol Python)

Terdapat banyak pilihan untuk berinteraksi dengan interpreter Python:

- Terminal default Python, dapat dijalankan dengan mengetikkan `python` pada terminal.
- IPython, merupakan interpreter Python dengan berbagai fitur tambahan seperti *tab-completion* dan *syntax highlighting.* IPython
- Jupyter qtconsole
- Jupyter notebook

Tampilan awal konsol Python default:

```
Python 3.6.6 |Anaconda custom (64-bit)| (default, Jun 28 2018, 17:14:51)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Tampilan awal IPython

```
Python 3.6.6 |Anaconda custom (64-bit)| (default, Jun 28 2018, 17:14:51)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

Aritmatika

```
>>> 2 + 3
5
>>> 2 / 3
0.6666666666666666
```

Variabel

```
>>> a = 3
>>> b = 4.1
>>> a*b
12.299999999999999
>>> a - b
-1.0999999999999996
>>> a/b
0.7317073170731708
```

4.1 Tipe data

Integer variables::

```
>>> 1 + 1
2
>>> a = 4
```

floats ::

```
>>> c = 2.1
```

Bilangan kompleks

```
>>> a = 1.5 + 0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

and booleans::

```
>>> 3 > 4
False
>>> test = (3 > 4)
>>> test
False
>>> type(test)
<type 'bool'>
```

A Python shell can therefore replace your pocket calculator, with the basic arithmetic operations +, -, *, /, % (modulo) natively implemented::

```
>>> 7 * 3.0
21.0
>>> 2**10
1024
>>> 8%3
2
```

Scalar types: int, float, complex, bool::

```
>>> type(1)
<type 'int'>
>>> type(1.)
<type 'float'>
>>> type(1. + 0j )
<type 'complex'>

>>> a = 3
>>> type(a)
<type 'int'>
```

* Type conversion::

»> float(1) 1.0

4.2 Containers

Python provides many efficient types of containers, in which collections of objects can be stored.

4.2.1 Lists

A list is an ordered collection of objects, that may have different types. For example ::

```
>>> l = [1, 2, 3, 4, 5]
>>> type(l)
<type 'list'>
```

Indexing: accessing individual objects contained in the list::

```
>>> l[2]
3
```

Counting from the end with negative indices::

```
>>> l[-1]
5
>>> l[-2]
4
```

warning

Indexing starts at 0** (as in C), not at 1 (as in Fortran or Matlab)!

Slicing: obtaining sublists of regularly-spaced elements

```
>>> l
[1, 2, 3, 4, 5]
>>> l[2:4]
[3, 4]
```

Warning

Note that l[start:stop] contains the elements with indices i such as `start<= i < stop` (i ranging from start to stop-1). Therefore, l[start:stop] has (stop-start) elements.

Slicing syntax: l[start:stop:stride]

All slicing parameters are optional::

```
>> l[3:]
4, 5]
>> l[:3]
1, 2, 3]
>> l[::2]
1, 3, 5]
```

Lists are mutable objects and can be modified::

```
>>> l[0] = 28
>>> l
[28, 2, 3, 4, 5]
>>> l[2:4] = [3, 8]
>>> l
[28, 2, 3, 8, 5]
```

Note;

The elements of a list may have different types::

```
>>> l = [3, 2, 'hello']
>>> l
[3, 2, 'hello']
>>> l[1], l[2]
(2, 'hello')
```

As the elements of a list can be of any type and size, accessing the i th element of a list has a complexity O(i). For collections of numerical data that all have the same type, it is **more efficient** to use the **array** type provided by the **Numpy** module, which is a sequence of regularly-spaced chunks of memory containing fixed-sized data istems. With Numpy arrays, accessing the i‘th‘ element has a complexity of O(1) because the elements are regularly spaced in memory.

Add and remove elements::

```
>>> l = [1, 2, 3, 4, 5]
>>> l.append(6)
>>> l
[1, 2, 3, 4, 5, 6]
>>> l.pop()
6
>>> l
[1, 2, 3, 4, 5]
>>> l.extend([6, 7]) # extend l, in-place
>>> l
[1, 2, 3, 4, 5, 6, 7]
>>> l = l[:-2]
>>> l
[1, 2, 3, 4, 5]
```

Reverse l

```
>>> r = l[::-1]
>>> r
[5, 4, 3, 2, 1]
```

Concatenate and repeat lists::

```
    >>> r + l
    [5, 4, 3, 2, 1, 1, 2, 3, 4, 5]
    >>> 2 * r
    [5, 4, 3, 2, 1, 5, 4, 3, 2, 1]
```

Sort r (in-place)::

```
>>> r.sort()
>>> r
[1, 2, 3, 4, 5]
```

Note:: **Methods and Object-Oriented Programming**

The notation r.method() (r.sort(), r.append(3), l.pop()) is our first example of object-oriented programming (OOP). Being a list, the object 'r' owns the *method* 'function' that is called using the notation **.**. No further knowledge of OOP than understanding the notation **.** is necessary for going through this tutorial.

4.3 Strings

Different string syntaxes (simple, double or triple quotes)::

```
s = 'Hello, how are you?'
s = "Hi, what's up"
s = '''Hello,
       how are you'''
s = """Hi,
   what's up?'''
```

The newline character is `\n`, and the tab characted is `\t`.

Strings are collections as lists. Hence they can be indexed and sliced, using the same syntax and rules.

Indexing::

```
    >>> a = "hello"
    >>> a[0]
    'h'
    >>> a[1]
    'e'
    >>> a[-1]
    'o'
```

(Remember that Negative indices correspond to counting from the right end.)

Slicing

```
>>> a = "hello, world!"
>>> a[3:6] # 3rd to 6th (excluded) elements: elements 3, 4, 5
'lo,'
>>> a[2:10:2] # Syntax: a[start:stop:step]
'lo o'
>>> a[::3] # every three characters, from beginning to end
'hl r!'
```

A string is an **immutable object** and it is not possible to modify its characters. One may however create new strings from an original one.

Strings have many useful methods, such as a.replace as seen above. Remember the a. object-oriented notation and use tab completion or help(str) to search for new methods.

String substitution

```
>>> 'An integer: %i; a float: %f; another string: %s' % (1, 0.1, 'string')
    'An integer: 1; a float: 0.100000; another string: string'
>>> i = 102
>>> filename = 'processing_of_dataset_%03d.txt'%i
>>> filename
'processing_of_dataset_102.txt'
```

4.4 Dictionary/Kamus

Tipe data kamus adalah tabel yang yang memetakan kunci ke suatu nilai.

is basically a hash table that **maps keys to values**. It is therefore an **unordered** container::

```
>>> tel = {'emmanuelle': 5752, 'sebastian': 5578}
>>> tel['francis'] = 5915
>>> tel
{'sebastian': 5578, 'francis': 5915, 'emmanuelle': 5752}
>>> tel['sebastian']
5578
>>> tel.keys()
['sebastian', 'francis', 'emmanuelle']
>>> tel.values()
[5578, 5915, 5752]
>>> 'francis' in tel
True
```

Suatu kamus dapat memiliki kunci dan nilai yang memiliki tipe berbeda

```
>>> d = {'a':1, 'b':2, 3:'hello'}
>>> d
{'a': 1, 3: 'hello', 'b': 2}
```

4.5 Tupel

Tuples are basically immutable lists. The elements of a tuple are written between brackets, or just separated by commas

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
```

```
(12345, 54321, 'hello!')
>>> u = (0, 2)
```

4.6 Set/Himpunan

Sets (himpunan): non ordered, unique items

```
>>> s = set(('a', 'b', 'c', 'a'))
>>> s
set(['a', 'c', 'b'])
>>> s.difference(('a', 'b'))
set(['c'])
```

Complex: cmath module

Boolean