Applied programming and numerical analysis

Lecture 1: Introduction and Programming of Python

Abstract

- Introduction
- Python
- Let's get started.

Homepage : http://www.ide.titech.ac.jp/~yamasita/APN/

1 Introduction

1.1 Purpose of this lecture

Department of Transdisciplinary Science and Engineering

- Problems has become more complex.
  - Increase of resource consumption and emissions
    ⇒ **Global environmental problems**
  - **Power of human beings has become strong comparing to nature in the earth.**
  - If the power is week, only a city or a culture will go to ruin. Such cases actually happened.
  - Because the power is stronger, human beings as well as other lives cannot survive.
- To solve such problems, international corporation and regulation are necessary.
- However, it is very difficult to solve such problems
  - Some groups of researchers say "No problem". (Even in our institute).
  - Solving the problems is postponed because it is not clear.
    (Untoward facts are neglected.)
  - **The Clash of Civilizations**

Engineering Transformation is necessary.

- **Department of General Medicine Primary Care**
  - When we go to a hospital, it is sometimes difficult to decide to which department we should go.
  - Patients do not concern about the department even surgery and internal medicine. What we need is only that the disease is cured.
  - However, now our destiny will be changed by the firstly selected department, surgery or internal medicine.
  - Furthermore, in case of a hospital of Japanese university, the first and second departments of internal medicine does not use the same curing procedure for the same disease.

- **Science and Engineering have be reconsidered**
  $\Rightarrow$ Transdisciplinary Science and Engineering
  - Consider not from chemical, machine, electronic, information, environmental engineerings but from the problem that should be solved.
  - Engineers tend to depend on their speciality.
  - Specialities such as civil and electrical engineerings have been split historically.
  - However, they may not be optimum.
  - There are many Overlaps.
  - Of course, an individual progress is important.
    (Robot, Hybrid car, solar power generation, HDTV)
  - **However, can they solve such huge problems?**

Transdeciplinary Science and Engineering

- **By getting over walls between countries or disciplines, it contributes to welfare of human beings.**

Programming and numerical analysis

- If we compare engineerings to functions in a man, computer is brain.
- That is very important to enhance intellectual activities of men.
- A Computer is not only in PC or a super computer but also in a smart phone, a camera, a car, a remote controller, etc.
- To make a thing, a design is necessary.
- The design should be evaluated to make a proper thing.
- If you make a real thing for the evaluation, it takes much cost or is sometimes dangerous.
- They should be virtually evaluated at first.
- For a complex thing, an analytic solution is not enough so that numerical analysis is necessary.
- Numerical analysis is used not only in chemical mechanical, electrical, computer, environmental and civil engineering but also in economics and

Programming and numerical analysis is a very **fundamental** subject in Transdisciplinary Science and Engineering.

1.2 Text book

1.3 Schedule (Yamashita)

1.	Guidance and introduction to Python (Yamashita)	12/4
	Programming of Python: Variables, expression, and control (Yamashita)	
2.	Programming of Python: Class (Yamashita)	12/11
	Practice: Sorting (Yamashita)	12/11
3.	Programming of Python: Array (Yamashita)	12/18
	Practice: Matrix calculation (Yamashita)	12/18
	Practice: Statistical calculation (Yamashita)	12/18
4.	Practice: Discrete Fourier transform	12/25
	Practice: Image processing	12/25

2 Introduction to Python

- Conceived in the late 1980s.
- Implemented in 1998.
- Python 2 was released in 2000.
- Python 3 was released in 2008.
  (We will use Python 3.4.)
- Python is a high-level programming language.
  - Low-level: similar to codes which CPUs execute directly.
    Examples: Machine language and Assembly language of which statement has almost one-to-one mapping to statement of the machine language.
  - High-level: easy to understand by humans
    Examples:FORTRAN, Java, and C++
- Python is general.
  - Targeted to an application domain.
    Examples: MATLAB for matrix calculation and R for statistical calculation.
  - General
    Examples: C, C++, Java, and Ruby Examples:FORTRAN, Java, and C++

- Python works by Interpreter.
  - Interpreter: Execute a line by a line of a source program.
    Example: JavaScript, PHP, and Ruby.
  - Compiler: A source program is converted to a program in machine language and the latter is executed in a computer.
    Example: C, C++, and FORTRAN

2.1 Let's get started

- We use "jupyter notebook" to execute a python program.
- Open a terminal and type:
- First we make a folder and move to it.

```
$ mkdir APN
$ cd APN
$ mkdir Python
$ cd Python
```

- Then, we start "jupyter notebook".

```
$ jupyter notebook
```

- A web browser starts and a cell to be input appears.
- Click 'New' and click 'Python 3'.
- A cell to be input appears.
- Please remember two short cut.
  - Ctrl-Enter (Push Ctrl key and Enter key simultaneously.): Execute command.
  - Shift-Enter (Push Shift key and Enter key simultaneously.): Make a new cell.
- Write the followings in a sell.

```
print("Hello world.")
```

- And type Ctrl-Enter.
- You can see `Hello world.`
- Type Shift-Enter and write

```
a = 4
b = 7
c = a + b
print(a, b, c)
```

- And type Ctrl-Enter.
- Rewrite the last line to

```
print("{0} + {1} = {2}".format(a, b, c))
```

- And type Ctrl-Enter.

3 Introduction to Python

3.1 Variables

Variables can contain a values, values with structure, and objects.
Identifier

- Name for a variable, a function, and a class.
- Letters can be used for name of variable:
  - Alphabet (`a`, `b`, ..., `A`, `B`, ...): Lower and upper cases are distinguished.)
  - Numeral (`0`, `1`, `2`, ...) They cannot be used for the beginning.
  - `_`
  - Almost all of Unicode (あ, ア, 阿, ...) Some of symbols are not allowed.
- Example
  - **Good**: `abc`, `_dAf_g`, エビシ, 阿, π
  - **NG**: `3abc`, `$abc`, 阿。 , 3 `abc`
- Keywords (Don't use as the name of variable.)
- `False`, `None`, `True`, `and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `nonlocal`, `not`, `or`, `pass`, `raise`, `return`, `try`, `while`, `with`, `yield`
- Reserved classes of identifiers
  `_*`, `__*`, `__*__`,

3.2 Type of data

- Every data is handled as a object in Python.
- Integer and float are also objects.

Embedded types for numbers

- `bool`
- `int`
- `float`
- `complex`

Embedded types for multiple data

- Immutable sequence
  - `tuple`
  - `string`
  - `bytes`
- Mutable sequence
  - `list`
  - `bytearray`

- Set
  - `set` (mutable)
  - `fronzenset` (immutable)
- Mapping
  - `dictionary`

3.3 Literal

A literal expresses a concrete value.

3.3.1 Numbers

- `bool` : `True`, `False`
- `int` : `123`, `-123`
- `float` : `2.5`, `-0.003`, `2.3e10`, `-2.553e-12`
- `complex` : `3.0+2.1j`, `-2.1e-2+3.2e3j`

3.3.2 None

`None`

3.3.3 String

- `'This is a pen.'`
- `"This is a pen."`

3.3.4 List

- `[1, 4, 2, 5, 1, -2]`
- `["This", "is", "a ", "pen"]`

3.3.5 Tuple

- `(1, 4, 2, 5, 1, -2)`
- `("This", "is", "a ", "pen")`
- `(1, "This", -3.0, "a")`

3.3.6 Dictionary

- `{1:"Freshman", 2:"Sophomore", 3:"Junior", 4:"Senior"}`
- `{"Freshman":1, "Sophomore":2, "Junior":3, "Senior":4}`
- `{(3, 4):7, (2, 4):"ABC", ("ab", 3)":112, ("dd", "ss"):"32"}`

3.3.7 Set

- `{1, 4, 2, 5, 1, -2}`
- `{"This", "is", "a ", "pen"}`
- `{1, "This", -3.0, "a"}`

3.4 Operator

3.4.1 Arithmetic operator

(Previlege Low → High)

Operator	meaning
`x + y`	
`x - y`	
`x * y`	
`x / y`	
`x // y`	devision as integers
`x % y`	
`-x`	
`+x`	
`x ** y`	x^y

3.4.2 Logical operator

(Previlege Low → High)

Operator	meaning
`x or y`	logical
`x and y`	logical and
`not x`	negation

3.4.3 Bit operator

(Previlege Low → High)

Operator	meaning
|	logical or for each bit
^	exclusive or for each bit
&	and for each bit
<<, >>	Bit shift (left, right)
~	Negation

3.4.4 Comparison operator

Operator	meaning
`x < y`	
`x <= y`	
`x > y`	
`x >= y`	
`x == y`	
`x != y`	
`x is y`	`x` and `y` are the same object.
`x is not y`	
`x in y`	`x` is included in `y`.
`x not in y`	

3.4.5 Membership operator

Operator	meaning
`x in y`	`x` is included in `y`.
`x not in y`	

3.4.6 Equality operator

Operator	meaning
`x is y`	`x` and `y` are the same object.
`x is not y`	

3.4.7 Cumulative assign operator

(Previlege Low → High)

Operator	meaning
`x += y`	`x = x + y`
`x -= y`	`x = x - y`
`x *= y`	`x = x * y`
`x /= y`	`x = x / y`
`x //= y`	`x = x // y`
`x %= y`	`x = x % y`
`x >>= y`	`x = x >> y`
`x <<= y`	`x = x << y`
`x &= y`	`x = x & y`
`x ^= y`	`x = x ^ y`
`x |= y`	`x = x | y`

3.4.8 Operators in Python

(Previlege Low → High)

Operator	meaning
`lambda`	
`if else`	
Logical operators	
Membership operators	
Equality operators	
Bit operators	(Except `~x`)
Arithmetic operators	(Except `+x`, `-x`, and `x ** y`)
`+x`, `-x`, and `~x`	
`x ** y`	
`x`.attribute	reference of attribute
`x[`index`]`, `x[`index`:`index`]`	indexces of array
`x(`expression, ... `)`	Call of a function

`(`expression, ... `)`
`[`expression, ... `]`
`[`key`:` value, ... `}`
`{`expression, ... `}`
	Tuple literal
	List literal
	Dictonary literal
	Set literal

3.5 Expression

- The expression can be evaluated and have a value.
- Examples:
  - literals: `3`, `[1, 2]`
  - Combination of operator and operand: `-x`, `x + y`, `z = x + y`, `x == y`
  - Function: `sin(x)`

3.6 Statement

- Statement expresses a procedure.
- Expression is also a statement.
- Examples of statement: `if`, `elif`, `else`, `break`, `continue`, and `import` statements.

3.7 Control

3.7.1 If

- Conditional execution.

```
x = 3
if (x == 3):
  print('x is three.')
print("End of program.")
```

3.7.2 else

```
x = 3
if (x == 3):
  print('x is three.')
else:
  print('x is not three.')
print("End of program.")
```

3.7.3 elif

```
x = 3
if (x == 3):
  print('x is three.')
elif (x == 7):
  print('x is seven.')
else:
  print('x is not three or seven.')
print("End of program.")
```

3.7.4 Nest

- Conditional sentences in a conditional sentence.
- Loop sentences in a loop sentence.

```
x = 3
y = 5
if (x == 3):
  if (y == 5):
    print('x is three and y is five.')
  else:
    print('x is three and y is not five.')
else:
  if (y == 9):
    print('x is not three and y is nine.')
  else:
    print('x is not three and y is not nine.')
  print('x is not three.')
print("End of program.")
```

3.7.5 for

- Loop sentence
- Range object : `range(2, 6)` is equivalent to `(2, 3, 4, 5)`

```
sum = 0
for x in range(1, 11):
  print(x)
  sum += x
print(sum)
```

By using list.

```
sum = 0
for x in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10):
  print(x)
  sum += x
print(sum)
```

Try by changing tuple to

- List: `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]` and `[2, 1, 3, 4, 5, 6, 7, 8, 9, 10]`
- Set: `{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}` and `{2, 1, 3, 4, 5, 6, 7, 8, 9, 10}`

```
dic = {1:"one one", 2:"two", 3:"surii"}
for key in dic:
  print(key)
  print(dic[key])
```

3.7.6 break, continue, else

- `break` : Exit from loop.
- `continue` : Latter part of loop is skipped.
- `else` : When the loop ended normally (not by `break`) the following block is executed.

```
sum = 0
for x in range(1, 11):
  print(x)
  if (x == 5):
    break
  sum += x
print(sum)
```

```
sum = 0
for x in range(1, 11):
  print(x)
  if (x == 5):
    continue
  sum += x
print(sum)
```

```
sum = 0
```

```
for x in range(1, 11):
  print(x)
  if (x == 5):
    break
  sum += x
else:
  print("Else sentence")
print(sum)
```

```
sum = 0
for x in range(1, 11):
  print(x)
  if (x == 5):
    continue
  sum += x
else:
  print("Else sentence")
print(sum)
```

3.7.7 while

- When the condition is true, the following block is executed.

```
sum = 0
x = 1;
while x < 11:
  print(x)
  sum += x
  x += 1
print(sum)
```

3.8 Function

- When you have many same processes for various values, it is not good to describe them respectively.
- Define a function that describes the process.
- A function is defined by `def`.
- A function of python can return multiple values by using `return`.
- When a function is called, arguments specified by order or variables.

Example

```
def printxy(x, y):
    print("x = {0}, y = {1}".format(x, y))

printxy(2, 4)
printxy(y = 2, x = 4)
```

Example

```
# Return product
def prod(x, y):
    prodv = x * y
    return prodv

a = 10
b = 7
u = prod(a, b)
print("{0} x {1} = {2}".format(a, b, u))
```

Example

```
# Euclidean algorithm
def euclid(x, y):
    u, v = x, y
    while(u != 0):
        u, v = (v % u), u
    else:
        lcd = v
        mcm = int(x * y / lcd)
    return lcd, mcm

x = 12
y = 9
a, b = euclid(x, y)
print("For {0} and {1}, LCD = {2}, MCM = {3}".format(x, y, a, b))
```

Value or reference.

```
def valOrRef(x, y):
    x = 11
    y[1] = 12
    print("x and y[1] in a function are {0} and {1}".format(x, y[1]))

x = 1
y = [1, 2, 3]
print("x and y[1] at first are {0} and {1}".format(x, y[1]))
valOrRef(x, y)
print("x and y[1] after the function are {0} and {1}".format(x, y[1]))
```

3.9 Report

- For every class, students have to submit a report in 7 days after the lecture.
- The file should be the nootbook format of ipython. Its file name should (student number)Lec(day of class).ipynb. It is `17B54321Lec1.ipynb` for example.
- Markdown cell is allowed to describe the report.
- Send the file by mail to `eniac1121@gmail.com` .

Markdown

- A blank line (Sometimes two blank lines) is not necessary to separate blocks.
- `#` : For titles
- `-` : For list. (Indent can be used)
- `1.` : For list with a number. (Indent can be used)
- (4 spaces or tab) For preformatted text (block).
- Two spaces after a text : New line
- Between two ‘ : For preformatted text in a line.
- Equation:

```
$$
\frac{1}{2} + \frac{1}{3} = \frac{5}{6}
$$
```

is displayed as

$$\frac{1}{2} + \frac{1}{3} = \frac{5}{6}$$

3.10 Objects

Discuss at the next class.