Pythonのプログラミング
Programming in Python

Pythonのプログラミング
Programming in Python

- Pythonはスクリプト言語である。
- Python is a scripting language.

- Pythonはオブジェクト指向言語である。
- Python is an object oriented language.

- Pythonは試作やデータ解析でよく使われる。
- Python is popular in prototyping and data analysis.

- If you want to study ruby, please visit https://www.codecademy.com/learn/learn-python/

- バージョン2.7系列と3系列がある。
- There are version 2.7 and 3

Pythonの実行環境
Execution environment of Python

- Pythonのプログラムを実行するにはいくつかの方法がある。
  - コマンドラインにプログラムを書く方法
  - 対話型で実行する方法
  - ファイルにプログラムを保存して実行する方法
  - Jupyter Notebookを使う方法

- There are several way to execute Python program.
  - Write a program in a command line
  - Interactive mode
  - Save a program to a file and execute it.
  - Use Jupyter notebook

- コマンドラインとしてプログラムを書く方法
- Write a program in a command line

```
% python3 -c 'print("Hello World!")'
```

  - 複数の文を1行に書きたい場合には「;」で繋げられる。
  - you can use ";" if you want to put multiple statement into one line.

Jupyter Notebook (Google Colaboratory)

- PythonのWebインターフェイス
- Web interface for Python
  - http://jupyter.org/

- Dockerなどを使ってもJupyter Notebookの環境を構築することができる。
- You can build an environment of Jupyter Notebook using Docker.

- Google Colaboratory (https://colab.research.google.com/)
  - ファイルの読み込みや書き出しが少し面倒
  - It is bit difficult to load and export.

Pythonの実行環境
Execution environment of Python

- エディタを使って、プログラムを入力する。どのエディタを使っても構いません。
- Write a program using an editor. You can use any text editor.

hello.py

```
1. #!/usr/bin/env python3
2. print("Hello World!")
```

```
% python3 hello.rb
```

- ファイルに実行権限をつければ、コマンドとしても実行できる。
- If you put execution permission to the file, you can execute the program as a command
  - 1行目で、どの様にコマンドを実行するかを指定している。この場合、"ruby filename"という形で実行される。
  - The first line specify how to execute a command. In this case, the command line is "ruby filename".

```
% chmod +x hello.py
% ./hello.py
```

Python入門
Introduction to Python

- `print( "Hello World!" );`

- データはオブジェクトとして扱われる。/ Data is stored as an Object.
- 文字列は、クオートかダブルクオートで括る。
  A string must be put in quotes or double quotes.

- Python3ではprintは関数と呼ばれる。
  "`print`" is called as "function" in python3
- Python2ではprintは文だった。
  "`print`" was statement in python2.7
- Usage: print(*objects, sep, end, file, flush)
- 似たような動作をするものに、`pprint`や`write`がある。
  There are similar function/method like "`pprint`" and "`write`".

- コメントは頭に#をつける。
  Put # in the beginning of line, if you want to put a comment

Python入門（変数と式）
Introduction to Python (variables and expressions)

- 変数 / Variables
  - 代入 / assignment: var = object

```
hello = "Hello World!"
print(hello)
```

  - 文字列をつなげる / concatenate strings

```
hello = "Hello World!"
sentence = "He said ¥"" + hello + "¥""
print(sentence)
```

  - 多重代入 / multiple assignment

```
a, b, c = 1, 2, "keio"
print(a, b, c, sep="¥n")
```

- 数値演算のために、+, -, *, /, %, //, (,)等を使うことができる。
  You can use +, -, *, /, %, //, (,) and so on for numeric operation.
- 整数か実数に注意 / Please pay attention to Integer or real number
  - Python3では割り切れないと自動的に実数になるが、Python2.7では整数になる。
  - In Python 3 it becomes a real number automatically if it is not divisible, but it becomes an integer in Python 2.7.

Python入門（条件分岐）
Introduction to Python (condition)

- if文 / if statement

```
if expression1 :
    procedure1
elif expression2 :
    procedure2
else:
    procedure3
```

  - Pythonではインデントがブロックを表す。
    In Python, indentation represents a block.

- Example

```
if x % 5 == 0:
    print("x is a multiple of 5!")
else:
    print("x is not a multiple of 5!")
```

Python入門（繰り返し）
Introduction to Python (loop)

- while文 / while statement

```
while expression :
    procedure
```

- for文 / for statement

```
for var in list :
    procedure
```

  - Example 1

```
l = [1, 2, 3, "keio" , "gijuku"]
for v in l:
    print(v)
```

  - Example 2

```
for v in range(10):
    print(v)
```

Python入門（繰り返し）
Introduction to Python (loop)

- continue文 / continue statement
  - continue文は次のループに移ることを指示する。
    The continue statement instructs to move to the next iteration.

- break文と繰り返しのelse節 / break statement and else clauses on loop
  - break文はloopを終了することを指示する。
    The break statement instructs to end the loop
  - 繰り返しのelse節は繰り返しを回りきった時は実行されるが、break文で途中終了した場合は実行されない。
    else clause on loop is executed when iteration is over, but it is not executed if ending by break statement.

```
for s in ["mita", "hiyoshi", "yagami", "shinanomachi", "fujisawa", "shiba"]:
    if s == "toyama":
          print("There is a campus of Keio university")
          break
else:
     print("There is no campus of Keio university")
```

Python入門（range関数）
Introduction to Python (range function)

- `range`関数は数字のリストを生成する。
  `range` function produces a list of numbers.

- `range(10)`
  0から10未満の整数を出力する。/ Outputs integer numbers from 0 to less than 10.

- `range(5, 10)`
  5から10未満の整数を出力する。/ Outputs integer numbers from 5 to less than 10.

- `range(0, 10, 3)`
  5から3ずつ増やしながら10未満の整数を出力する。 / Outputs integer numbers from 5 to less than 10 while incrementing by 3.

- 以下の違いに注意。/ Please note followings differences
  - `print(range(10))`
  - `print(list(range(10)))`

Python入門（演習）
Introduction to Python (exercise)

- 次に示すプログラムを実行するとなにが出力されるか。
- What do you get when the following program runs.

ex01.py

```
#!/usr/bin/env python
for i in range(1,100) :
    if i % 3 == 0 :
        continue
    print(i)
```

ex02.py

```
#!/usr/bin/env python
for i in range(1,100) :
    if i % 3 == 0 :
        break
    print(i)
```

ex03.rb

```
#!/usr/bin/env python
a = ["keio", "university", "sfc"]
for s in a :
    print(s)
```

ex04.rb

```
#!/usr/bin/env python
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print(n, 'equals', x, '*', n//x)
            break
    else:
        print(n, 'is a prime number')
```

Python入門（モジュール）
Introduction to Python (modules)

- Pythonでは他の人が作ったモジュール（プログラム）をインポートすることができる。
- Python can import modules (program) that are made by other users.

```
import sys
print(sys.argv)
```

- モジュールの参照名を変更することもできる。
- You can change the reference name of the module.

```
import sys as s
print(s.argv)
```

- pandas, numpy, scikit-learn などデータサイエンティストに有名なモジュールが多数存在する。
- There are many famous modules for data scientists such as pandas, numpy and scikit-learn.

Python入門（Pandasのデータフレーム）
Introduction to Python (DataFrame of Pandas)

- データフレームの生成 / Creation of dataframe

```
import pandas as pd
m = [[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
     [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
     [20, 21, 22, 23, 24, 25, 26, 27, 28, 29]]
df = pd.DataFrame(m, columns=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'])
```

- 列の抽出 / get specific columns

```
print(df['A'])
print(df.B)
```

- 行の抽出 / get specific lines

```
print(df[1:2])
print(df[:2])
```

  - 0行目から始まるので注意。/ Please note that start from 0

- 条件を指定した行の抽出 / get specific lines that fit to the condition

```
print(df[df['A'] > 5])
print(df[df.B > 5])
```

Python入門（CSV）
Introduction to Python (CSV)

- Comma Separated Value

- PythonでCSVを扱うためにはpandasを使うのがお勧め。
- It is recommended to use pandas to handle CSV in Python

- CSVファイル全体の読み込み / read whole CSV file at once

```
import pandas as pd
df = pd.read_csv("hoge.csv")
```

- CSVを標準入力から読む/ read CSV file from stdin

```
import sys
import pandas as pd
df = pd.read_csv(sys.stdin)
```

- データフレームをCSVにする / convert DataFrame to CSV

```
print(df.to_csv())
```

Rubyのプログラミング
Programming in Ruby

Rubyのプログラミング
Programming in Ruby

- Rubyはスクリプト言語である。
- Ruby is a scripting language.

- Rubyはオブジェクト指向言語である。
- Ruby is an object oriented language.

- Rubyは試作やウェブプログラミングでよく使われる。
- Ruby is popular in prototyping and Web programming.

- If you want to study ruby, please visit
  https://www.codecademy.com/learn/ruby

Rubyでの初めてのプログラム
First program in Ruby

- Rubyのプログラムを実行するには2つの方法がある。（本当は、3つあるが1つはここでは扱わない。）
  - コマンドラインにプログラムを書く方法
  - ファイルにプログラムを保存して実行する方法
- There are two (three actually, but we don't use one of them) way to execute Ruby program.
  - Write a program in a command line
  - Save a program to a file and execute it.

- コマンドラインとしてプログラムを書く方法
- Write a program in a command line

```
% ruby -e 'print("Hello World!¥n")'
```

- 複数の文を1行に書きたい場合には「;」で繋げられる。
- you can use ";" if you want to put multiple statement into one line.

Rubyでの初めてのプログラム
First program in Ruby

- エディタを使って、プログラムを入力する。どのエディタを使っても構いません。
- Write a program using an editor. You can use any text editor.

hello.rb

```
1.#!/usr/bin/env ruby
2.print("Hello World!¥n")
```

```
% ruby hello.rb
```

- ファイルに実行権限をつければ、コマンドとしても実行できる。
- If you put execution permission to the file, you can execute the program as a command
  - 1行目で、どの様にコマンドを実行するかを指定している。この場合、"ruby filename"という形で実行される。
  - The first line specify how to execute a command. In this case, the command line is "ruby filename".

```
% chmod +x hello.rb
% ./hello.rb
```

Ruby入門
Introduction to Ruby

- `print( "Hello World!¥n" );`

- データはオブジェクトとして扱われる。/ Data is stored as an Object.
- 文字列は、クオートかダブルクオートで括る。
  A string must be put in quotes or double quotes.

- printはメソッドと呼ばれる。/ "`print`" is called as "method".
- STDOUT.print()とも書ける。/ You can write as "STDOUT.print()".
- 次に示すように括弧は省略することができる。
  You can omit "()" as follows.

```
print "Hello World!¥n"
```

- 似たようなメソッドに、`puts`や`p`がある。
  There are similar methods like "`puts`" and "`p`".

- コメントは頭に#をつける。
  Put # in the beginning of line, if you want to put a comment

Ruby入門（変数と式）
Introduction to Ruby (variables and expressions)

- 変数 / Variables
  - 代入 / assignment: var = object

```
hello = "Hello World!"
puts(hello)
```

  - 文字列の中に変数を入れる / puts variables into a string
    - #{var}

```
hello = "Hello World!"
sentence = "He said ¥"#{hello}¥""
puts(sentence)
```

  - 文字列をつなげる / concatenate strings

```
hello = "Hello World!”
sentence = "He said ¥"" + hello + "¥""
puts(sentence)
```

  - 多重代入 / multiple assignment

```
a, b, c = 1, 2, "keio"
```

- 数値演算のために、+, -, *, /, %, (,)等を使うことができる。
  You can use +, -, *, /, %, (,) and so on for numeric operation.

Ruby入門（条件分岐）
Introduction to Ruby (condition)

- if文 / if statement

```
if expression1 then
    procedure1
elsif expression2 then
    procedure2
else
    procedure3
end
```

  - "then"は省略可能だが、読みやすさのためには省略しないほうが良い。
    "then" can be omitted. But you shouldn't do for readability.

- if修飾子 / if modifier

```
puts("Good morning World!") if hour < 12;
```

Ruby入門（繰り返し）
Introduction to Ruby (loop)

- while文 / while statement

```
while expression do
    procedure
end
```

- times文 / times statement

```
num.times do
    procedure
end
```

- for文 / for statement

```
for var in start..end do
    procedure
end
```

- "do"は省略可能。 / "do" can be omitted.
- "do〜end"は"{}"で代替可能。/ "do-end" can be replaced by "{}".

```
% ruby -e "l = STDIN.readlines; 10000.times {print l.sample}" < 73.csv
```

Ruby入門（配列と繰り返し、ループの制御）
Introduction to Ruby (Array, loop and control)

- 配列 / Array

```
a = [1, 2, "keio"]
a = [1, ["keio, "sfc"], ["big", "data"]]
```

- 配列を使った繰り返し / Loop with array

```
array.each do |var|
    p var
end

for var in array do
    p var
end
```

  - '(1..10)' といった表記は、1〜10までの [1, 2, 3, ..., 10] のような配列を作る。
    '(1..10)' makes an array from 1 to 10, i.e. [1, 2, 3, ..., 10]

- ループの制御 / control of loop
  - break:　ループを終了する / exit the loop
  - next:　次の回に飛ぶ / jump to next turn

Ruby入門（演習）
Introduction to Ruby (exercise)

- 次に示すプログラムを実行するとなにが出力されるか。
- What do you get when the following program runs.

ex01.rb
```
#!/usr/bin/env ruby
for i in 1..100 do
    next if i%3 == 0
    p i
end
```

ex02.rb
```
#!/usr/bin/env ruby
for i in 1..100 do
    break if i%3 == 0
    p i
end
```

ex03.rb
```
#!/usr/bin/env ruby
a = ["keio", "university", "sfc"]
a.each do |var|
    print(var + "¥n")
end
```

ex04.rb
```
#!/usr/bin/env ruby
while line = gets() do
  line.chomp!();
  puts("Hello, #{line}.")
end
```

- chomp: 文字列の最後の改行を削った値を返す。
- chomp!: 文字列から最後の改行を削る。
- chomp: return the string without newline.
- chomp!: omit newline from the string

Ruby入門（コマンドライン引数）
Introduction to Ruby (command line parameter)

- プログラムで、コマンドライン引数を使うことができる。
  The program can use command line parameters.

```
% ./hello.rb taro
```

- 組み込み変数のARGVを使う / Use built-in variable, ARGV
- ARGVは配列 / ARGV is an array.
- 1つめのパラメータはARGV[0]
  first parameter can be referred by using ARGV[0]
- 複数のパラメータがある場合は、順に配列に入る。
  If there are multiple parameters, these are stored in array in order.

argv.rb

```
#!/usr/bin/env ruby
ARGV.each do |param|
  p param
end
```

Ruby入門（CSV）
Introduction to Ruby (CSV)

- Comma Separated Value

- RubyはCSVのための入出力インターフェイスを持っている。
- Ruby has CSV input/output interface

- CSVファイル全体の読み込み / read whole CSV file at once

```
data = CSV.read(filename)
```

- CSVファイルから1行ずつ読み込み / read line by line from CSV file

```
CSV.foreach(filename) do |row|
    p row
end
```

  - 上記で変数rowは配列になっている。
    Variable `row` is an array in above example.

Ruby入門（CSV）
Introduction to Ruby (CSV)

- 標準入力からCSVを読み込む。/ Read CSV from standard input

```
CSV(STDIN).each do |row|
    p row
end
```

- CSVをファイルから読み込む(2)。/ Read CSV from a file (2).

```
f = open(ARGV[0])
CSV(f).each do | row |
  p row
end
```

前処理における定石（1）
Formula of preprocessing (1)

CSV

- ファイルを1度に読み込んで、1レコードずつ処理をする
  Read whole file, process line by line

csv_all_data.rb

```
1.  #!/usr/bin/env ruby
2.  require 'csv'
3.
4.  data = CSV.read(ARGV[0])
5.  for row in data
6.  #  p row
7.  end
```

- ファイルを1行1行処理する / Read line by line

csv_line_by_line.rb

```
1.  #!/usr/bin/env ruby
2.  require 'csv'
3.
4.  data = CSV.open(ARGV[0])
5.  for row in data
6.  #  p row
7.  end
```

- 処理時間とメモリ使用量 / Processing time and used memory size

```
% ./csv_all_data.rb 5.csv
535872kB, 16.621702s for reading, 0.082764s for loop
% ./csv_line_by_line.rb 5.csv
 14632kB, 11.247075s
```

前処理における定石（2）
Formula of preprocessing (2)

時系列データ(日毎) / Time series data

- 一度にファイルを読み込む
  Read a file at once

time_all_data.rb

```
1.  #!/usr/bin/env ruby
2.  require 'csv'
3.  require 'time'
4.  require 'date'
5.
6.  data = CSV.read(ARGV[0])
7.  startDate = Date.parse(ARGV[1])
8.  endDate = Date.parse(ARGV[2])
9.
10. for day in startDate..endDate
11.   for row in data
12.     next if Time.parse(row[0]).to_date < day
13.     next if Time.parse(row[0]).to_date > day
14. #   p row
15.   end
16. end
```

  - Used memory: 1,530,176 kB
  - メモリに全てのデータが読み込まれているとはいえ、2重ループのため、時間がかかる。
    It takes long time, even all data are loaded onto memory. Because there are dual loops.

- 1行ずつ処理する
  Process line by line

time_line_by_line.rb

```
1.  #!/usr/bin/env ruby
2.  require 'csv'
3.  require 'time'
4.  require 'date'
5.
6.  data = CSV.open(ARGV[0])
7.  startDate = Date.parse(ARGV[1])
8.  endDate = Date.parse(ARGV[2])
9.
10. for day in startDate..endDate
11.   for row in data
12.     next if Time.parse(row[0]).to_date < day
13.     break if Time.parse(row[0]).to_date > day
14. #   p row
15.   end
16. end
```

  - Used memory: 18,548 kB
  - レコードが時刻順に並んでいることが前提。
    Records must be in order.
  - 上記のプログラムは、1日の初めの1行を捨てている。本当はその処理が必要。
    In the program, 1st line of a day is omitted.