A Crash Course in Python

The Basics

Python 설치하기 2.7 ver.

- Anaconda / IPython 권장

The Zen of Python

Whitespace Formatting

In [ ]:

```
for i in [1, 2, 3, 4, 5]:
        print i # first line in "for i" block
        for j in [1, 2, 3, 4, 5]:
            print j # first line in "for j" block
            print i + j # last line in "for j" block
        print i # last line in "for i" block
    print "done looping"
```

In [2]:

```
long_winded_computation = (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11 + 12 + 13 + 14 + 15
+ 16 + 17 + 18 + 19 + 20)
```

In [ ]:

```
  - and for making code easier to read:
```

In [3]:

```
list_of_lists = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
easier_to_read_list_of_lists = [ [1, 2, 3],
                                 [4, 5, 6],
                                 [7, 8, 9] ]
```

- blackslash /

In [ ]:

```
two_plus_three = 2 + ₩
                 3
```

Modules

- Python에서 작성한 스크립트나 프로그램을 Import 하여 모듈 형태로 사용할 수 있다.

In [ ]:

```
import re
my_regex = re.compile("[0-9]+", re.I)
```

In [ ]:

```
import re as regex
my_regex = regex.compile("[0-9]+", regex.I) # 모듈 이름이 길 경우, as 축약
```

In [ ]:

```
import matplotlib.pyplot as plt
```

In [ ]:

```
from collections import defaultdict, Counter
lookup = defaultdict(int)
my_counter = Counter()
```

In [ ]:

```
### Arithmetic
```

In [4]:

```
5/2   #정수로만 값을 낸다.
```

Out[4]:

2

In [7]:

```
from __future__ import division
5/2   #
```

Out[7]:

2.5

In [8]:

```
5//2
```

Out[8]:

2

Functions

In [ ]:

```
def double(x):
    return x * 2
```

In [ ]:

```
def apply_to_one(f):
    """calls the function f with 1 as its argument"""
    return f(1)

my_double = double # refers to the previously defined function
x = apply_to_one(my_double) # equals 2
```

In [ ]:

```
It is also easy to create short anonymous functions, or lambdas:

y = apply_to_one(lambda x: x + 4) # equals 5
```

In [9]:

```
def subtract(a=0, b=0):                #2개 이상인 경우
    return a - b
```

In [10]:

```
subtract(10, 5) # returns 5
```

Out[10]:

5

String

- '','',\
- """ : 문자열이 길 때

Exceptions

In [ ]:

```
try:
    print 0 / 0
except ZeroDivisionError:
    print "cannot divide by zero"
```

List

- 1차원 순차 자료형
- [], list 함수 :

In [ ]:

```
integer_list = [1, 2, 3]
heterogeneous_list = ["string", 0.1, True]
list_of_lists = [ integer_list, heterogeneous_list, [] ]
list_length = len(integer_list) # equals 3
list_sum = sum(integer_list) # equals 6
```

In [ ]:

```
x = range(10) # is the list [0, 1, ..., 9]
zero = x[0] # equals 0, lists are 0-indexed
one = x[1] # equals 1
nine = x[-1] # equals 9, 'Pythonic' for last element
eight = x[-2] # equals 8, 'Pythonic' for next-to-last element
x[0] = -1 # now x is [-1, 1, 2, 3, ..., 9]          #리스트는 원소의 수정이 가능하다
```

In [ ]:

```
first_three = x[:3] # [-1, 1, 2]
three_to_end = x[3:] # [3, 4, ..., 9]
one_to_four = x[1:5] # [1, 2, 3, 4]
last_three = x[-3:] # [7, 8, 9]
without_first_and_last = x[1:-1] # [1, 2, ..., 8]
copy_of_x = x[:] # [-1, 1, 2, ..., 9]
```

In [ ]:

```
1 in [1, 2, 3] # True
0 in [1, 2, 3] # False
```

In [13]:

```
x = [1, 2, 3]
x.extend([4, 5, 6]) # x is now [1,2,3,4,5,6]
```

In [14]:

```
x
```

Out[14]:

```
[1, 2, 3, 4, 5, 6]
```

In [15]:

```
x = [1, 2, 3]
y = x + [4, 5, 6] # y is [1, 2, 3, 4, 5, 6]; x is unchanged
```

In [16]:

```
y
```

Out[16]:

```
[1, 2, 3, 4, 5, 6]
```

In [18]:

```
x.append(0) # x is now [1, 2, 3, 0]
```

In [19]:

```
x
```

Out[19]:

```
[1, 2, 3, 0]
```

In [ ]:

```
y = x[-1] # equals 0
z = len(x) # equals 4
```

In [ ]:

```
### Tuples
```

In [ ]:

```
my_list = [1, 2]
my_tuple = (1, 2)
other_tuple = 3, 4
my_list[1] = 3 # my_list is now [1, 3]
```

In [ ]:

```
try:
    my_tuple[1] = 3
except TypeError:
    print "cannot modify a tuple"  #수정 불가하므로 error 발생
```

- convenient way to return multiple values from functions:

In [20]:

```
def sum_and_product(x, y):
    return (x + y),(x * y)
```

In [22]:

```
sp = sum_and_product(2, 3) # equals (5, 6)
```

In [23]:

```
sp
```

Out[23]:

```
(5, 6)
```

In [ ]:

```
Tuples (and lists) can also be used for multiple assignment:

x, y = 1, 2 # now x is 1, y is 2
x, y = y, x # Pythonic way to swap variables; now x is 2, y is 1
```

Dictionaries

- key를 바탕으로 value를 찾는다.
- {}

In [ ]:

```
empty_dict = {} # Pythonic
empty_dict2 = dict() # less Pythonic
grades = { "Joel" : 80, "Tim" : 95 } # dictionary literal
```

In [ ]:

```
But you'll get a KeyError if you ask for a key that's not in the dictionary:  #없는 키 값을 요청할 경우 error

try:
kates_grade = grades["Kate"]
except KeyError:
print "no grade for Kate!"
```

In [ ]:

```
You can check for the existence of a key using in:  # key의 존재 여부를 체크함


joel_has_grade = "Joel" in grades # True
kate_has_grade = "Kate" in grades # False
```

In [ ]:

```
you look up a key that's not in the dictionary:

joels_grade = grades.get("Joel", 0) # equals 80
kates_grade = grades.get("Kate", 0) # equals 0
no_ones_grade = grades.get("No One") # default default is None
```

In [ ]:

```
You assign key-value pairs using the same square brackets:

grades["Tim"] = 99 # replaces the old value
grades["Kate"] = 100 # adds a third entry
num_students = len(grades) # equals 3
```

In [ ]:

```
We will frequently use dictionaries as a simple way to represent structured data:

tweet = {
"user" : "joelgrus",
"text" : "Data Science is Awesome",
"retweet_count" : 100,
"hashtags" : ["#data", "#science", "#datascience", "#awesome", "#yolo"]
}

#딕셔너리를 쓰면, 하나의 구조형태로 여러 값을 표현할 수 있다.
```

-defaultdict

In [ ]:

```

```

-Counter

In [ ]:

```
  from collections import Counter
  c = Counter([0, 1, 2, 0]) # c is (basically) { 0 : 2, 1 : 1, 2 : 1 }
```

Sets

- 중복된 값을 허용하지 않는 자료형.
- Unordered

In [25]:

```
  s = set()
```

In [26]:

```
s.add(1) # s is now { 1 }
```

In [27]:

```
s.add(2) # s is now { 1, 2 }
```

In [28]:

```
s.add(2) # s is still { 1, 2 }
```

In [29]:

```
x = len(s) # equals 2
```

In [30]:

```
y = 2 in s # equals True
```

In [31]:

```
z = 3 in s # equals False
```

item_list = [1, 2, 3, 1, 2, 3] num_items = len(item_list) # 6 item_set = set(item_list) # {1, 2, 3} num_distinct_items = len(item_set) # 3 distinct_item_list = list(item_set) # [1, 2, 3]

Control Flow

In [ ]:

```
if 1 > 2:
    message = "if only 1 were greater than two..."
elif 1 > 3:
    message = "elif stands for 'else if'"
else:
    message = "when all else fails use else (if you want to)"
```

- 반복문 : while, for

In [ ]:

```
#while loop:

x = 0
while x < 10:
    print x, "is less than 10"
    x += 1
```

In [ ]:

```
#for and in:

for x in range(10):
    print x, "is less than 10"
```

In [ ]:

```
#continue and break:

for x in range(10):
    if x == 3:
        continue # go immediately to the next iteration
    if x == 5:
        break # quit the loop entirely
    print x


#This will print 0, 1, 2, and 4.
```

Truthiness

In [ ]:

```
one_is_less_than_two = 1 < 2 # equals True
true_equals_false = True == False # equals False
```

In [ ]:

```
Python uses the value None to indicate a nonexistent value. It is similar to other languag
es'
null:

x = None
print x == None # prints True, but is not Pythonic
print x is None # prints True, and is Pythonic
```

In [ ]:

```
Python lets you use any value where it expects a Boolean. The following are all
 "Falsy":
The Basics | 25
• False
• None
• [] (an empty list)
• {} (an empty dict)
• ""
• set()
• 0
• 0.0
```

NOT SO BASIC

Sorting

In [ ]:

```
x = [4,1,2,3]
y = sorted(x) # is [1,2,3,4], x is unchanged
x.sort() # now x is [1,2,3,4]
```

In [ ]:

```
sorted from largest to smallest, you can specify a reverse=True

# sort the list by absolute value from largest to smallest
x = sorted([-4,1,-2,3], key=abs, reverse=True) # is [-4,3,-2,1]

# sort the words and counts from highest count to lowest
wc = sorted(word_counts.items(),
key=lambda (word, count): count,
reverse=True)
```

List Comprehensions

- 기존의 list 객체를 이용해 조합
- 필터링 등의 추가적인 연산을 통해 새로운 객체를 생성하는 경우

In [ ]:

```
even_numbers = [x for x in range(5) if x % 2 == 0] # [0, 2, 4]
squares = [x * x for x in range(5)] # [0, 1, 4, 9, 16]
even_squares = [x * x for x in even_numbers] # [0, 4, 16]
```

In [ ]:

```
similarly turn lists into dictionaries or sets:

square_dict = { x : x * x for x in range(5) } # { 0:0, 1:1, 2:4, 3:9, 4:16 }
square_set = { x * x for x in [1, -1] } # { 1 }
```

In [ ]:

```
A list comprehension can include multiple fors:

pairs = [(x, y)
for x in range(10)
for y in range(10)] # 100 pairs (0,0) (0,1) ... (9,8), (9,9)
```

In [ ]:

```
and later fors can use the results of earlier ones:

increasing_pairs = [(x, y) # only pairs with x < y,
for x in range(10) # range(lo, hi) equals
for y in range(x + 1, 10)] # [lo, lo + 1, ..., hi - 1]
```

Generators and Iterators

- List, Tuple, Strings 처럼 순회 가능한 객체에는 Iterator 라는 특별한 객체가 포함되어 있다.
- Generator는 Iterator를 만드는 강력한 도구

- yield operator : One way to create generators is with functions and the yield operator

In [ ]:

```
def lazy_range(n):
"""a lazy version of range"""
i = 0
while i < n:
yield i
i += 1
```

Randomness

In [ ]:

```
we will frequently need to generate random numbers, which we can do with the random modul
e:

import random
```

In [ ]:

```
randomly reorders the elements of a list:

up_to_ten = range(10)
random.shuffle(up_to_ten)
print up_to_ten
# [2, 5, 1, 9, 7, 3, 8, 6, 4, 0] (your results will probably be different)
```

In [ ]:

```
If you need to randomly pick one element from a list you can use random.choice:

my_best_friend = random.choice(["Alice", "Bob", "Charlie"]) # "Bob" for me
```

Regular Expressions

- Regular expressions provide a way of searching text

In [ ]:

```
import re

print all([ # all of these are true, because
        not re.match("a", "cat"), # * 'cat' doesn't start with 'a'
        re.search("a", "cat"), # * 'cat' has an 'a' in it
        not re.search("c", "dog"), # * 'dog' doesn't have a 'c' in it
        3 == len(re.split("[ab]", "carbs")), # * split on a or b to ['c','r','s']
        "R-D-" == re.sub("[0-9]", "-", "R2D2") # * replace digits with dashes
           ]) # prints True
```

Object-Oriented Programming

Functional Tools

Enumerate

In [ ]:

```
# not Pythonic
for i in range(len(documents)):
document = documents[i]
do_something(i, document)
```

In [ ]:

```
# also not Pythonic
i = 0
for document in documents:
do_something(i, document)
i += 1
```

In [ ]:

```
The Pythonic solution is enumerate, which produces tuples (index, element):

for i, document in enumerate(documents):
do_something(i, document)
```

In [ ]:

```
Similarly, if we just want the indexes:

for i in range(len(documents)): do_something(i) # not Pythonic
for i, _ in enumerate(documents): do_something(i) # Pythonic

We'll use this a lot.
```

zip and Argument Unpacking

- we will need to zip two or more lists together
- zop : transforms multiple lists into a single list of tuples of corresponding elements

In [ ]:

```
list1 = ['a', 'b', 'c']
list2 = [1, 2, 3]
zip(list1, list2) # is [('a', 1), ('b', 2), ('c', 3)]
```

In [ ]:

```
“unzip” a list using a strange trick:

pairs = [('a', 1), ('b', 2), ('c', 3)]
letters, numbers = zip(*pairs)
```

args and kwargs

In [ ]:

```
  - create a higher-order function
     that takes as input some function f and returns a new function that for any input re
turns twice the value of f
```

In [ ]:

```
def doubler(f):
    def g(x):
        return 2 * f(x)
    return g
```

In [ ]:

```
This works in some cases:

def f1(x):
    return x + 1

    z`g = doubler(f1)
    print g(3) # 8 (== ( 3 + 1) * 2)
    print g(-1) # 0 (== (-1 + 1) * 2)
```