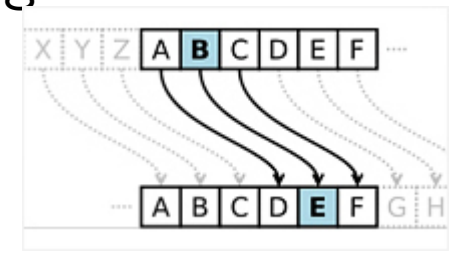
Lecture #13 | 모듈/패키지, 예외처리

SE213 프로그래밍 (2019)

Written by Mingyu Cho
Edited by Donghoon Shin

4th assignment (6/3 midnight)

- 과제 4-1: 인코딩/디코딩



- **주의사항**: 반드시 **재귀호출**을 이용하여 구현할 것
- encode(key_list, plaintext) → (0~25) 정수를 원소로 갖는 키 리스트와 평문을 받아서 암호문을 반환
- decode(key_list, ciphertext) → 암호에 사용된 키 리스트와 암호문을 받아서 평문을 반환
- 암호화 방식
  - 매 소문자 마다 암호키 리스트를 이용하여 암호화
  - 현재 글자를 포함하여 미변환 문자의 길이를 암호 키의 길이로 나눈 나머지 값을 키의 인덱스로 활용
  - print(encode([3,0,1], "hello world!"))
  - kflop zproe! -〉 hhmlr wrslg!! -〉 ieomo xoumd!!!

4th assignment (6/3 midnight)

- 과제 4-2: MyCounter 확장
  - __init__(self, value=0) → 초기화 (카운터 개수)
  - number_of_counters() → 만들어진 카운트 개수 반환
  - get(self) → 현재 카운터 객체의 value값 반환
  - inc(self, value=1) → 현재 카운터 객체의 값을 value만큼 증가
  - dec(self, value=1) → 현재 카운터 객체의 값을 value만큼 감소
  - stats(self) → 카운터 객체의 초기값, 증가 누적수, 감소 누적수를 튜플로 반환

4th assignment (6/3 midnight)

- 과제 4-3: 수업 관리 클래스 (ClasManager)
  - __init__(self, student_list, course_title="") → 학생 리스트를 받아서 초기화
  - get_student(self, ids) → id 값을 받으면, 학생리스트에서 해당 학생 객체 반환 (없으면 문자열반환)
  - sort(self, column="id") → 현재 객체가 가지고 있는 학생리스트를 column에 따라서 정렬

  - (참고) print_all(self) → 현재 학생리스트를 포맷에 맞춰 출력
- 학생 클래스(Student)
  - 클래스 변수(info="DGIST", ids=2018001
  - __init__(self, name, mid, final, assi) : 학번을 포함하여 초기화
  - get_info(self): 현재 객체의 각종 정보를 문자열로 리턴
  - get_total(self): 점수 반영비율을 넣어서 계산한 총점 반환
  - get_grade(self): 현재 총점을 기준으로 학점 반환
  - __eq__(st1, ids): 학생 객체와 학번 또는 또다른 학생 객체를 받아서 동일 여부 반환
  - __repr__(self) : 객체의 공식 문자열로 학번을 반환

지난 시간에 다룬 내용

- 정렬 알고리즘
  - Selection Sort
  - Merge Sort
  - QuickSort

오늘 다룰 내용

- 모듈
- 함수/클래스/모듈의 문서화
  - docstring
  - `help()`
  - `dir()`
- 예외처리

모듈/패키지 사용하기

- 모듈을 읽어들이는 방법
  - import module_name
  - module_name은 .py을 뺀 파일이름
- 모듈 사용방법: module_name.xxx 와 같이 사용하면 모듈에 속한 함수 혹은 변수 이름을 뜻한다. 사용 예:
  - module_name.function_name()
  - module_name.variable_name

예제: math

```
# using math
import math
# constants
print(math.pi)
print(math.e)
# power and logarithmic functions
x = 1.0
print(math.exp(x))   # return e ** x
print(math.log(x))   # return natural log(x)
print(math.log10(x)) # return log(x) with base 10
print(math.sqrt(x))  # return square root(x)
# triangular functions
print(math.cos(x))   # return cos(x)
print(math.sin(x))   # return sin(x)
```

```
3.141592653589793
2.718281828459045
2.718281828459045
0.0
0.0
1.0
0.5403023058681398
0.8414709848078965
```

import 하는 여러 가지 방법

- 일반적인 방법

```
import math
print(math.pi, math.e)
```

- 특정 객체(변수, 함수, 클래스 등)만 import

```
from math import pi, e
print(pi, e)
```

- 모든 객체를 import

```
from math import *
print(pi, e, tau)
```

  – 사용자가 정의한 변수, 함수, 클래스와 혼동이 될 수 있으므로, 권장되지 않음

import 하는 여러 가지 방법

- 모듈 이름이 간략히 줄임: 관례적인 경우에 따르는 것이 좋음

```
import math as m
print(m.pi, m.e)
```

- 모듈 이름을 줄여서 사용하는 경우는 관례적인 경우만 따르는 것이 좋음, 예:

```
import numpy as np
import pandas as pd
```

```
math.pi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'math' is not defined
```

모듈을 만드는 방법과 import의 의미

- 어떤 파이썬 파일도 모듈로 사용이 가능함
  - .py를 제외한 파일 이름이 모듈 이름이 됨
  - 파이썬 파일 안에서 정의한 변수, 함수, 클래스를 import해서 사용할 수 있음

- 모듈을 import 할 때, 수행되는 작업
  - 모듈을 위한 별도의 네임스페이스가 생성됨
  - 파이썬 프로그램을 한 줄씩 읽으면서 실행함
    - 변수, 함수, 클래스 등이, 모듈을 위해 만들어진 네임스페이스 안에 생성됨
    - 화면 출력 등과 같은 입출력 명령도 실행됨

예시: 모듈의 생성 및 사용

```python
#vector.py
def add(v1, v2):
    v = []
    for i in range(len(v1)):
        v.append(v1[i] + v2[i])
    return v
```

```python
#main.py
import vector

t1 = [42, 1024, 23]
t2 = [6, 28, 496]
print(vector.add(t1, t2))
```

```
[48, 1052, 519]
```

__name__

- __name__: 파이썬 프로그램이 어떻게 실행되고 있는지를 나타내는 전역 변수
  - 개별적으로 실행될 때 (예: elice, pycharm, shell 등에서 실행): '__main__'
  - 모듈로 import 되어 실행될 때: 모듈 이름
- 참고: 아래와 같은 if문을 작성해서 import되지 않을 때만 수행되는 테스트 코드 등을 넣는 것이 일반적임

```
if __name__ == '__main__':
    # example code or test code
    # ...
```

docstring, help(), dir()

- docstring: 클래스, 함수 등의 설명을 적은 것
  - 클래스나 함수의 선언문 바로 뒤에 여러 줄에 걸친 문자열(""" 로 시작과 끝을 표시)을 의미함
  - 참고: https://www.python.org/dev/peps/pep-0257/

- help(): 클래스에 대한 설명을 출력 (docstring 포함)

- dir(): 클래스의 속성(함수, 변수)을 모두 출력

예시: docstring, help(), dir()

```
class MyCounter:
    """MyCounter is a simple implementation of a counter"""
    def __init__(self, value=0):
        self.counter = 0

    def inc(self):
        """Increase counter value by 1"""
        self.counter += 1

    def get(self):
        """Return the current counter value"""
        return self.counter

print('Help on MyCounter')
help(MyCounter)
print('Help on MyCounter,inc()')
help(MyCounter.inc)
print('dir() on MyCounter')
print(dir(MyCounter))
```

실행 결과: docstring, help(), dir() (1/3)

```
Help on MyCounter
      1 Help on class MyCounter in module __main__:
      2
      3 class MyCounter(builtins.object)
      4  |  MyCounter is a simple implementation of a counter
      5  |
      6  |  Methods defined here:
      7  |
      8  |  __init__(self, value=0)
      9  |      Initialize self.  See help(type(self)) for accurate
signature.
     10  |
     11  |  get(self)
     12  |      Return the current counter value
     13  |
     14  |  inc(self)
     15  |      Increase counter value by 1
     16  |
```

실행 결과: docstring, help(), dir() (2/3)

```
     17  |  ----------------------------------------------------------------
-------
     18  |  Data descriptors defined here:
     19  |
     20  |  __dict__
     21  |      dictionary for instance variables (if defined)
     22  |
     23  |  __weakref__
     24  |      list of weak references to the object (if defined)

Help on MyCounter,inc()
      1 Help on function inc in module __main__:
      2
      3 inc(self)
      4     Increase counter value by 1
```

실행 결과: docstring, help(), dir() (3/3)

```
dir() on MyCounter
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__le__', '__lt__',
'__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '__weakref__', 'get', 'inc']
```

* 사용자가 정의하지 않아도 기본적으로 정의되는 method들이 여러 개 있음

예시: fibonacci.py

```
""" Provides Fibonacci related functions"""

def fibonacci_list(n):
    """Return Fibonacci series up to n"""
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result

if __name__ == '__main__':
    assert(fibonacci_list(10) == [1, 1, 2, 3, 5, 8])
```

Build-in moudules

```
import time
now = time.localtime()

s = "%04d-%02d-%02d %02d:%02d:%02d" % (now.tm_year,
now.tm_mon, now.tm_mday, now.tm_hour, now.tm_min,
now.tm_sec)


print(time.time())
```

```
2019-05-28 07:29:47
1558996215.1387784
```

Error types

- **Syntax Error**
  - print “test” (python 3.x)
  - var =
  - var * =) + !@#
  - if True
    print(“True”)
- **Runtime Error**
  - Zero division
  - 1 + int(input())
- **Logical Error**
  - Hard to find

```
 File "<stdin>", line 1
   var * = )
         ^
SyntaxError: invalid syntax


Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Exception Handling

- 예외 처리 구문
- try:

```
        # 예외가 발생할 수 있는 명령문들의 코드 블록
except  예외이름 :
        # 해당 예외 발생시, 처리하는 코드 블록
else:
        # 예외가 발생하지 않을 때 수행되는 코드 블록
finally:
        # 예외 발생 여부와 관계없이 항상 수행되는 코드 블록
```

Python built-in Exceptions

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      |    +-- ModuleNotFoundError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      |    +-- RecursionError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
      +-- SystemError
      +-- TypeError
      +-- ValueError
      |    +-- UnicodeError
      |         +-- UnicodeDecodeError
      |         +-- UnicodeEncodeError
      |         +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
           +-- ResourceWarning
```

Exception Handling(예: name error)

- NameError: name 'math' is not defined

```
import math as m

print(math.pi)

class Student:
    idn = 20180001
    def __init__(self):
        self.idn = idn
```

```
import math as m

try:
    print(math.pi)
except:
    print("Exception")

class Student:
    idn = 20180001
    def __init__(self):
        self.idn = idn
try:
    st1 = Student()
except NameError:
    print("NameError")
```

Exception Handling(예: index error)

- IndexError: list index out of range

```
lst = list(range(10))
lst.remove(3)


print(lst[0])
print(lst[9])
```

```
lst = list(range(10))
lst.remove(3)

try:
    print(lst[9])
except IndexError:
    print("index error")

try:
    print(lst[10])
except LookupError:
    print("index error")
```

Exception Handling(예: value error)

- ValueError: -1 is not in list

```
lst = list(range(10))
lst.remove(3)


print(lst.index(5))
print(lst.index(-1))
```

```
lst = list(range(10))
lst.remove(3)

try:
    print(lst.index(-1))
except IndexError:
    print("index error")

try:
    print(lst.index(-1))
except ValueError:
    print("value error")
```

Exception Handling(예: key error)

- KeyError: 'a'

```
01: def count_items(sequence):
02:     count = {}
03:     for item in sequence:
04:         count[item] += 1
05:     return count
06:
07: t = ['a', 'b', 'a', 'c', 'a']
08: print(count_items(t))
```

```
01: def count_items(sequence):
02:     count = {}
03:     for item in sequence:
04:         count[item] =
count.get(item, 0) + 1
05:     return count
06:
07: t = ['a', 'b', 'a', 'c', 'a']
08: print(count_items(t))
```

Multiple Exception Handling

```
base = 100
while True:
    print(base/int(input()))
```

```
base = 100
while True:
    # Value Error
    try:
        print(base/int(input()))
    except ValueError:
        pass
    except:
        pass
```

```
base = 100
while True:
    # Value Error
    try:
        print(base/int(input()))
    except ValueError:
        pass
    except ZeroDivisionError:
        pass
```

```
base = 100
while True:
    # Value Error
    try:
        print(base/int(input()))
    except (ValueError, ZeroDivisionError):
        pass
```

Exception Handling - finally, else

```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("division by zero!")
    else:
        print("result is", result)
    finally:
        print("executing finally clause")
divide(2, 1)
print()
divide(2, 0)
print()
divide("2", "1")
```

```
result is 2.0
executing finally clause

division by zero!
executing finally clause

executing finally clause
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s)
for /: 'str' and 'str '
```

ANY QUESTIONS?