# 알고리즘 실습

180913 - Python 이해
Data types, Flow control, Function, Class

# 오늘의 목표

- Data types
- Flow control
- Function
- Class

# Data types

# Data types

- 데이터의 형태: 숫자, 문자, 참거짓, ...
- 우리는 현실에서 자료형을 `자연스럽게' 인지함
    - ex) 학생이 10명 있어!: 10은 숫자
- 컴퓨터에서는 어떻게 구별할까?
- 왜 구별해야할까?

# Python data types

- 10개 내외로 95%를 사용함
- 자주 사용하는 것만 배울 예정
- 나머지는 필요할 때 찾아서 활용

- 4. Built-in Types
  - 4.1. Truth Value Testing
  - 4.2. Boolean Operations — `and`, `or`, `not`
  - 4.3. Comparisons
  - 4.4. Numeric Types — `int`, `float`, `complex`
  - 4.5. Iterator Types
  - 4.6. Sequence Types — `list`, `tuple`, `range`
  - 4.7. Text Sequence Type — `str`
  - 4.8. Binary Sequence Types — `bytes`, `bytearray`, `memoryview`
  - 4.9. Set Types — `set`, `frozenset`
  - 4.10. Mapping Types — `dict`
  - 4.11. Context Manager Types
  - 4.12. Other Built-in Types
  - 4.13. Special Attributes

# Boolean types

- 참/거짓을 나타내는 타입
- True / False

# Boolean operations

- Boolean type에만 적용할 수 있는 연산
- and, or, not

# String type

- Python에서 문자열은 3가지 표현 방법이 있음: 내부 표현 방식은 같음
  - 'String'
  - “String”
  - ‘’’Also string’’’, “””Also string”””

```
20  text1 = '홑따옴표로 묶으면 문자열'
21  text2 = "쌍따옴표로 묶어도 '문자열'"
22  text3 = '''홑따옴표 3개로 묶어도
23  "문"
24  "자"
25  "열"'''
26  text4 = """쌍따옴표 3개로 묶어도
27  '문' '자' '열'"""
28
29  print(text1)
30  print(text2)
31  print(text3)
32  print(text4)
```

# String operations

- 수 많은 연산이 존재
  - 간단한 것만 소개
- str.upper(): 대문자로 만들어서 반환
- str.lower(): 소문자로 만들어서 반환
- str[x:y]: x에서 y번째 글자만 잘라내기

# Numeric types

- 수를 나타내는 자료형
- int: 정수형
  - int()
- float: 소수형
  - float()

```
34  user_num = input('숫자를 입력하세요. ')
35  print('입력한 값의 자료형: {0}'.format(type(user_num)))
36
37  float_num = float(user_num)
38  print('변환된 값: {0}'.format(float_num))
39  print('변환된 값의 자료형: {0}'.format(type(float_num)))
40
41  int_num = int(float_num)
42  print('변환된 값: {0}'.format(int_num))
43  print('변환된 값의 자료형: {0}'.format(type(int_num)))
```

```
숫자를 입력하세요. 10.4
입력한 값의 자료형: <class 'str'>
변환된 값: 10.4
변환된 값의 자료형: <class 'float'>
변환된 값: 10
변환된 값의 자료형: <class 'int'>
```

# Numeric operations

- 4칙 연산 및 편리한 연산이 존재
- 2개의 숫자를 입력받아서 각 연산 결과를 출력하는 프로그램

| Operation | Result |
|---|---|
| `x + y` | sum of *x* and *y* |
| `x - y` | difference of *x* and *y* |
| `x * y` | product of *x* and *y* |
| `x / y` | quotient of *x* and *y* |
| `x // y` | floored quotient of *x* and *y* |
| `x % y` | remainder of `x / y` |
| `-x` | *x* negated |
| `+x` | *x* unchanged |
| `abs(x)` | absolute value or magnitude of *x* |
| `int(x)` | *x* converted to integer |
| `float(x)` | *x* converted to floating point |
| `x ** y` | *x* to the power *y* |

# Sequence types

- 순서가 있는 자료`들'
- list(), tuple(), range()
  - [‘일번', '이번', '삼번', '일번'] -> [‘일번', '이번', '삼번', '일번', ‘이번']
  - (3, 2, 1)
  - range(0, 10, 2)

# list() operations

- list.append()
  - 원소 추가
- list.remove()
  - 원소 제거
- 같은 원소를 넣으면 어떻게 될까?

```
45  var_list = list()
46  print(var_list)
47  var_list.append('문자열')
48  print(var_list)
49  var_list.append(913)
50  print(var_list)
51  var_list.append('문자열')
52  print(var_list)
53  var_list.append(True)
54  print(var_list)
55  var_list.pop(0)
56  print(var_list)
57  var_list.remove(True)
58  print(var_list)
```

```
[]
['문자열']
['문자열', 913]
['문자열', 913, '문자열']
['문자열', 913, '문자열', True]
[913, '문자열', True]
[913, '문자열']
```

# Set type

- 순서가 없는 자료
- 집합이라고도 표현

# Set operations

- set.add()
  - 원소 추가
- set.remove()
  - 원소 제거
- 같은 원소를 넣으면 어떻게 될까?

```
60  var_set = set()
61  print(var_set)
62  var_set.add('문자열')
63  print(var_set)
64  var_set.add(913)
65  print(var_set)
66  var_set.add('문자열')
67  print(var_set)
68  var_set.add(True)
69  print(var_set)
70  var_set.remove(True)
71  print(var_set)
```

```
set()
{'문자열'}
{913, '문자열'}
{913, '문자열'}
{913, True, '문자열'}
{913, '문자열'}
```

# Mapping type

- 다른 언어와 차별화되는 Python의 가장 편리한 자료형
  - 많은 언어에서 없거나 사용하기 어렵게 구현되어 있음
- Key-Value 형태로 저장되는 자료

# Mapping operation

- 참조를 사용하여 추가
- dict.pop()을 사용하여 제거
- 중복된 원소가 추가되면 어떻게 될까?
- 순서가 있을까?

```
73  var_dict = dict()
74  var_dict['Key'] = 'Value'
75  print(var_dict)
76  var_dict[9] = 13
77  print(var_dict)
78  var_dict[(4, 2, 1)] = {'키-튜플', '밸류-셋'}
79  print(var_dict)
80  var_dict.pop('Key')
81  print(var_dict)
```

```
{'Key': 'Value'}
{'Key': 'Value', 9: 13}
{'Key': 'Value', 9: 13, (4, 2, 1): {'밸류-셋', '키-튜플'}}
{9: 13, (4, 2, 1): {'밸류-셋', '키-튜플'}}
```

# 여기서 잠깐! 파이썬 코딩 컨벤션!

- 파이썬 코딩에 관한 가이드 라인이 존재: PEP 8
  - 탭 대신 스페이스 4개를 사용
  - 1줄에 79자가 넘지 않도록 코딩
  - 함수와 클래스, 코드 블럭 사이에 빈 줄로 구분
  - 가능하면 주석
  - 함수 주석 사용
  - 콤마, 연산자 다음 스페이스 사용
  - 클래스, 함수 이름 일관성을 지켜서 사용: 클래스(MyClass), 함수/메소드(my_function)

# Flow control

# 소스코드 수행 순서

- Python에서는 가장 윗 줄부터 차례대로 수행됨
- 변수를 사용할 땐 그 위에서(사용하기 전) 변수가 할당(선언)되어야 함
    - 할당: 데이터를 저장하기 위해 메모리 공간을 마련하는 것

# 참고) 주석

- 대부분의 프로그래밍 언어에서는 `주석'을 지원함
- 주석: 소스코드에는 있으나 컴퓨터가 실행하지 않는 부분
- 예시:
  - in C: // This is comment, /* This is multi line comments */
  - in Python: # This one line comment, ‘’’This is multiple line comments’’’

# if … elif … else

- `조건'을 확인하고 `분기': 조건에 따라서 수행하는 부분이 변함
- if … elif … else 활용

```
num = 43
if num % 2 == 0:
    print('짝수')
elif num % 2 == 1:
    print('홀수')
else:
    print('소수')
```

# Comparisons

- 대부분의 프로그래밍 언어에서는 `참' 혹은 `거짓' 나타내는 방법이 있음
  - Python: True / False
  - C: not 0 / 0
  - JAVA: true / false
- 다양한 데이터를 비교할 수 있음
  - How?
  - __lt__(), __eq__()

| Operation | Meaning |
|---|---|
| `<` | strictly less than |
| `<=` | less than or equal |
| `>` | strictly greater than |
| `>=` | greater than or equal |
| `==` | equal |
| `!=` | not equal |
| `is` | object identity |
| `is not` | negated object identity |

# Boolean Operation

- 비교문의 결과 (참/거짓)를 조합해야하는 경우가 있음

| Operation |
| --- |
| `x or y` |
| `x and y` |
| `not x` |

  - And! Or! Not!

| A | B | A and B | A or B | not A |
| --- | --- | --- | --- | --- |
| True | True | True | True | False |
| True | False | False | True | False |
| False | True | False | True | True |
| False | False | False | False | True |

# while ...

- 소스코드의 일정 부분을 반복하고 싶을 때!
- 얼마만큼 반복하게 되는가?
  - 공백 4개!
  - Block
- 조건이 언제나 참이라면?

```
91    num = 59
92    while num > 0:
93        print('현재 숫자 {0}! 2로 나눕니다.'.format(num))
94        num = num // 2
```

# for ...

- 일정 횟수만 반복하고 싶을 때
- for ...
- range(start, end, [step])

```
96  for index in range(0, 10):
97      print('range()를 활용한 index! {0}'.format(index))
98
99  var_list = ['이렇게', '활용할', '수', '있음!', '편리함']
100 for var in var_list:
101     print(var)
```

# Function

# 함수 생성

- def: 함수를 정의하는 키워드
- https://docs.python.org/3/reference/compound_stmts.html#function-definitions

```
>>> def fib(n):    # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print(a, end=' ')
...         a, b = b, a+b
...     print()
...
>>> # Now call the function we just defined:
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

# Class

# 클래스 생성

- class: 클래스 생성 키워드
- https://docs.python.org/3/tutorial/classes.html

```
class MyClass:
    """A simple example class"""
    i = 12345

    def f(self):
        return 'hello world'
```

# 질문이 생기면?

- 이름: 문현수
- 전공: 통신및보안
- 과정: 석박사통합과정 8학기
- 연구실: 데이터네트워크연구실(공5633)
- 메일: munhyunsu@cs-cnu.org
- 알고리즘은 함께 해결해가는 과목이므로 과감하게 연락
- 이메일로 처리가 안 되는 급한일: 문자/전화 등