프로그래밍

이은섭
(Eunseop Lee)

Computer Science and Engineering,
Pohang University of Science and Technology
eseop90@gmail.ac.kr

변수와 자료형

변수

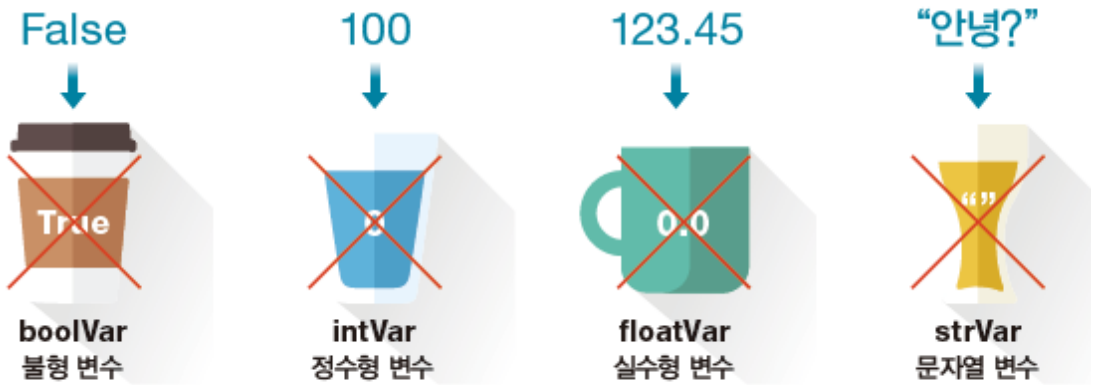
- 변수는 어떠한 값을 저장하는 메모리 공간
- 변수 선언
  - 메모리 공간을 준비하는 것
  - '변수이름 = 값'

```
>>> a = 1
>>> b = "python"
```

변수

- 변수는 값을 담으면(대입하면) 사용 가능. 변수에 있던 기존 값은 없어지고 새로 입력한 값으로 변경됨


```
boolVar = False
intVar = 100
floatVar = 123.45
strVar = "안녕?"
```



변수

- 변수에는 변수의 값을 넣을 수도 있고, 계산 결과를 넣을 수도 있음

```
var2 = 200
var1 = var2
```


```
var1 = 100 + 100
```

변수

- 변수에 저장되는 '값'에 따라 변수의 자료형이 결정됨
  - 가장 많이 사용되는 자료형: 불형(Boolean), 정수형, 실수형, 문자형

- type() 함수를 사용하여 변수의 자료형을 확인할 수 있음

```
boolVar = True
intVar = 0
floatVar = 0.0
strVar = ""

type(boolVar), type(intVar), type(floatVar), type(strVar)
```

출력 결과

```
(<class 'bool'>, <class 'int'>, <class 'float'>, <class 'str'>)
```

변수명 규칙

- 변수명의 기본 규칙은 영문 대소문자, 밑줄(_), 숫자를 조합해서 사용
- 특수문자(!@#$%^&*)나 공백 ' '은 변수명에 사용할 수 없음
- 숫자로 시작할 수 없음
- 예약어는 변수명으로 사용할 수 없음
  - 예약어: True, False, None, and, or, not, break, continue, return, if, else, elif, for, while, except, finally, global, import, try
- 대/소문자를 구분(myVar와 MyVar는 다른 변수)

변수명 규칙

- 함수 이름, 변수 이름 및 파일 이름은 설명적이어야 하며, 축약어를 사용하지 마십시오. 특히 프로젝트 외부에서 독자가 모호하거나 생소한 축약어를 사용하지 않으며, 단어 내에서 글자를 삭제하여 축약하지 않습니다.
- Function names, variable names, and filenames should be descriptive; avoid abbreviation. In particular, do not use abbreviations that are ambiguous or unfamiliar to readers outside your project, and do not abbreviate by deleting letters within a word. (Google Styles Guidelines)

https://google.github.io/styleguide/pyguide.html

불형

- 불(bool) 자료형이란 참(True)와 거짓(False)을 나타내는 자료형

```
>>> a = True
>>> b = False
```

```
>>> type(a)
<class 'bool'>
>>> type(b)
<class 'bool'>
```

불형

- 1==1은 '1과 1이 같은가?'를 묻는 조건이며, 1과 1은 같으므로 True를 리턴함

```
>>> 1 == 1
True
```

- 2는 1보다 크므로 2<1조건문은 거짓, 즉 False를 리턴함

```
>>> 2 < 1
False
```

불형

- 자료형에 하나의 값이 있을 경우 참(True), 비었을 경우 거짓(False)를 리턴함
- 조건문에서 주로 사용됨

```
>>> if []:
...     print("참")
... else:
...     print("거짓")
...
거짓
```

```
>>> if [1, 2, 3]:
...     print("참")
... else:
...     print("거짓")
...
참
```

값	참 or 거짓
"python"	참
""	거짓
[1, 2, 3]	참
[]	거짓
(1, 2, 3)	참
()	거짓
{'a': 1}	참
{}	거짓
1	참
0	거짓
None	거짓

불형

- 반복문에 사용되는 조건문

```
>>> a = [1, 2, 3, 4]
>>> while a:
...     print(a.pop())
...
4
3
2
1
```

```
while 조건문:
    수행할_문장
```

불형

- bool() 함수를 사용하면 자료형의 참과 거짓을 식별할 수 있음

```
>>> bool('python')
True
```

```
>>> bool('')
False
```

숫자형

- 숫자형(Number)이란 숫자 형태로 이루어진 자료형

- 정수형(Integer)
  - 정수를 뜻하는 자료형

```
>>> a = 123
>>> a = -178
>>> a = 0
```

- 실수형(Floating-point)
  - 소수점이 포함된 숫자

```
>>> a = 1.2
>>> a = -3.45
```

숫자형

- 실수형(Floating-point)
  - 컴퓨터식 지수 표현 방식

```
>>> a = 4.24E10
>>> a = 4.24e-10
```

  - 여기서 4.24E10은 4.24×10^{10}, 4.24e-10은 4.24×10^{-10}를 의미함

- 2진수
  - 2진수를 만들기 위해서는 숫자 0 + 알파벳 b(소문자) 또는 B(대문자)로 시작

```
>>> a=0b101010
>>> print(a)
42
```

  - $0b101010 = 1\times2^5 + 1\times2^3 + 1\times2^1 = 42$

숫자형

- 8진수
  - 8진수를 만들기 위해서는 숫자 0 + 알파벳 o(소문자) 또는 O(대문자)로 시작

```
>>> a = 0o177
>>> print(a)
127
```

  - $0o177 = 1\times8^2 + 7\times8^1 + 7 = 127$

- 16진수
  - 16진수를 만들기 위해서는 숫자 0 + 알파벳 x(소문자) 또는 X(대문자)로 시작

```
>>> a = 0x8ff
>>> b = 0xABC
>>> print(b)
2748
```

  - $0xABC = 10\times16^2 + 11\times16^1 + 12 = 2748$

숫자형을 활용하기 위한 연산자

- 사칙연산

```
>>> a = 3
>>> b = 4
>>> a + b
7
>>> a - b
-1
>>> a * b
12
>>> a / b
0.75
```

숫자형을 활용하기 위한 연산자

- **연산자
  - x**y로 사용했을 때, x의 y제곱(x^y)값을 리턴함

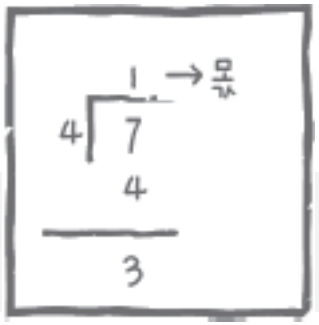
```
>>> a = 3
>>> b = 4
>>> a ** b
81
```

- /연산자
  - 나눗셈 연산자

```
>>> 7 / 4
1.75
```

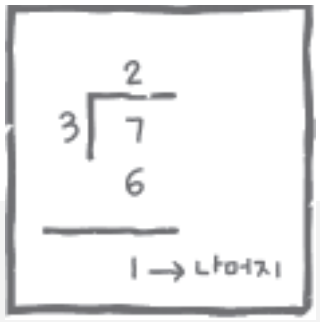
숫자형을 활용하기 위한 연산자

- //연산자
  - //는 나눗셈의 몫을 리턴하는 연산자




```
>>> 7 // 4
1
```

- %연산자
  - %는 나눗셈의 나머지 값을 리턴하는 연산자




```
>>> 7 % 3
1
>>> 3 % 7
3
```

숫자형을 활용하기 위한 연산자

- 복합 연산자
  - a에 1이라는 값이 대입되어 있을 때, a의 값을 1만큼 증가시키려면?

```
>>> a = a + 1  # a + 1로 계산된 값을 다시 a에 대입한다.
>>> print(a)
2
```

  - 하지만 복합 연산자인 +=를 사용하여 더 간략하게 표현될 수 있음

```
>>> a += 1
```

  - 만약 a의 값을 1만큼 감소 시키려면 -= 복합 연산자를 사용할 수 있음

```
>>> a = 1
>>> a -= 1  # a = a - 1 과 같음
>>> print(a)
0
```

  - 이 외에도, *=, /=, //=, %=, **=가 존재

문자열

1. 큰따옴표로 양쪽 둘러싸기
   - "Hello World"
2. 작은따옴표로 양쪽 둘러싸기
   - 'Python is fun'
3. 큰따옴표 3개를 연속으로 써서 양쪽 둘러싸기
   - """Life is too short, you need python"""
4. 작은따옴표 3개를 연속으로 써서 양쪽 둘러싸기
   - '''Life is too short, You need python'''

- 문자열을 만드는 방법은 왜 4가지일까?

문자열

- 작은따옴표가 사용될 때
  - Python's favorite food is perl 문자열을 변수에 저장하려면?

```
>>> food = "Python's favorite food is perl"
```

- 큰따옴표가 사용될 때
  - “Python is very easy.” he says. 문자열을 변수에 저장하려면?

```
>>> say = '"Python is very easy." he says.'
```

- 역슬러시를 사용하여 작은따옴표와 큰따옴표를 문자열에 표현하기

```
>>> food = 'Python\'s favorite food is perl'
>>> say = "\"Python is very easy.\" he says."
```

문자열

- 문자열에 개행 추가하기
  - 아래 문자열을 변수에 대입하려면 어떻게 해야할까?

```
Life is too short
You need python
```

  - 이스케이프 코드(₩n)

```
>>> multiline = "Life is too short\nYou need python"
```

  - 연속된 작은따옴표 3개 또는 큰따옴표 3개

```
>>> multiline='''
... Life is too short
... You need python
... '''
```

```
>>> multiline="""
... Life is too short
... You need python
... """
```

이스케이프(escape) 코드

- 프로그래밍할 때 사용할 수 있도록 미리 정의해 둔 '문자 조합'
- 주로 출력물을 보기 좋게 정렬하는 용도로 사용함

코드	설명
`\n`	문자열 안에서 줄을 바꿀 때 사용
`\t`	문자열 사이에 탭 간격을 줄 때 사용
`\\`	`\`를 그대로 표현할 때 사용
`\'`	작은따옴표(')를 그대로 표현할 때 사용
`\"`	큰따옴표(")를 그대로 표현할 때 사용
`\r`	캐리지 리턴(줄 바꿈 문자, 커서를 현재 줄의 가장 앞으로 이동)
`\f`	폼 피드(줄 바꿈 문자, 커서를 현재 줄의 다음 줄로 이동)
`\a`	벨 소리(출력할 때 PC 스피커에서 '삑' 소리가 난다)
`\b`	백 스페이스
`\000`	널 문자

문자열 연산

- 문자열 더하기

```
>>> head = "Python"
>>> tail = " is fun!"
>>> head + tail
'Python is fun!'
```

- 문자열 곱하기

```
>>> a = "python"
>>> a * 2
'pythonpython'
```

문자열 길이

- 'len()' 함수를 사용하여 문자열의 길이를 구할 수 있음
- 문자열의 길이에는 공백 문자도 포함됨

```
>>> a = "Life is too short"
>>> len(a)
17
```

문자열 인덱싱과 슬라이싱

- 문자열 인덱싱
  - 인덱싱(indexing)이란 무엇을 '가르킨다'의 의미를 가짐

```
>>> a = "Life is too short, You need Python"
```

L	i	f	e		i	s		t	o	o		s	h	o	r	t	,		Y	o	u		n	e	e	d		P	y	t	h	o	n
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	30	31	32	33

  - 위 문자열에서 'L'은 첫 번째 자리를 뜻하는 숫자 0, 'i'는 1, 중간에 있는 short의 's'는 12가 '가르킨다'
  - 문자를 가르키는 번호를 인덱스(index)라 하며, 해당 인덱스의 문자를 호출하는 것을 인덱싱이라 함

```
>>> a = "Life is too short, You need Python"
>>> a[3]
'e'
```

문자열 인덱싱과 슬라이싱

- 인덱스 앞에 마이너스(-)를 붙이면 문자열 뒤에서부터 인덱싱함

```
>>> a = "Life is too short, You need Python"
>>> a[0]
'L'
>>> a[12]
's'
>>> a[-1]
'n'
>>> a[-2]
'o'
>>> a[-5]
'y'
```

문자열 인덱싱과 슬라이싱

- 문자열 슬라이딩
  - 슬라이싱(slicing)은 무엇인가를 '잘라 낸다'의 의미를 가짐
  - "Life is too short, You need Python" 문자열에서 'Life' 또는 'You'와 같은 단어를 뽑아내는 방법이 없을까?
  - 변수[시작 인덱스 : 끝 인덱스]

```
>>> a = "Life is too short, You need Python"
>>> a[0:4]
'Life'
```

문자열 인덱싱과 슬라이싱

- 문자열 슬라이딩 예시
  - a[4]는 공백 문자이기 때문에 'Life'가 아닌 'Life_'가 출력됨

```
>>> a[0:5]
'Life '
```

  - 시작 인덱스는 꼭 0일 필요는 없음

```
>>> a[0:2]
'Li'
>>> a[5:7]
'is'
>>> a[12:17]
'short'
```

  - 시작 인덱스를 생략하면 문자열의 처음부터 끝 인덱스까지 잘라냄

```
>>> a[19:]
'You need Python'
```

문자열 인덱싱과 슬라이싱

- 문자열 슬라이딩 예시
  - 시작 인덱스와 끝 인덱스를 생략하면 전체 문자열을 잘라냄

```
>>> a[:]
'Life is too short, You need Python'
```

  - 슬라이싱에서도 마이너스(-) 기호를 사용할 수 있음
  - a[19:-7]=a[19:27] (len(a)=34)

```
>>> a[19:-7]
'You need'
```

문자열 인덱싱과 슬라이싱

- 부분 반환 문자열에서 문자 사이의 간격(step)을 정할 수 있음
  - str[start:end:step]

```
>>> a = 'python'
>>> a[::1]
'python'
>>> a[1:5:2]
'yh'
>>> a[1:5:3]
'yo'
```

  - step은 음수도 가능하며, 이 경우 역순으로 구성된 문자열을 반환

```
>>> a = 'python'
>>> a[::-1]
'nohtyp'
>>> a[5:0:-1]
'nohty'
>>> a[-1:-7:-1]
'nohtyp'
```

문자열 포매팅

- 온도 측정 프로그램을 작성했다고 가정해 보자.
  - "현재 온도는 18도입니다."
- 시간이 지나서 20도가 되면 다음 문장을 출력한다.
  - "현재 온도는 20도입니다."
- 두 문자열은 숫자를 제외하고 모두 같다.

- 문자열 안에 특정한 값을 바꿔야 할 경우, 문자열 포맷팅(string formatting)을 사용함

문자열 포매팅

- 숫자 대입
  - 숫자를 넣고 싶은 자리에 %d 문자를 넣어 주고 삽입할 숫자는 % 다음에 작성
  - %d는 '문자열 포맷 코드'라고 부른다.

```
>>> "I eat %d apples." % 3
'I eat 3 apples.'
```

- 문자열 대입
  - 문자열 포맷 코드로 %s를 사용

```
>>> "I eat %s apples." % "five"
'I eat five apples.'
```

코드	설명
%s	문자열(String)
%c	문자 1개(character)
%d	정수(Integer)
%f	부동소수(floating-point)
%o	8진수
%x	16진수
%%	Literal % (문자 % 자체)

```
>>> "Error is %d%%." % 98
'Error is 98%.'
```

문자열 포매팅

- 변수 대입
  - 변수의 자료형에 따라 문자열 포맷 코드를 사용함

```
>>> number = 3
>>> "I eat %d apples." % number
'I eat 3 apples.'
```

```
>>> a = 'five'
>>> "I eat %s apples." %a
'I eat five apples.'
```

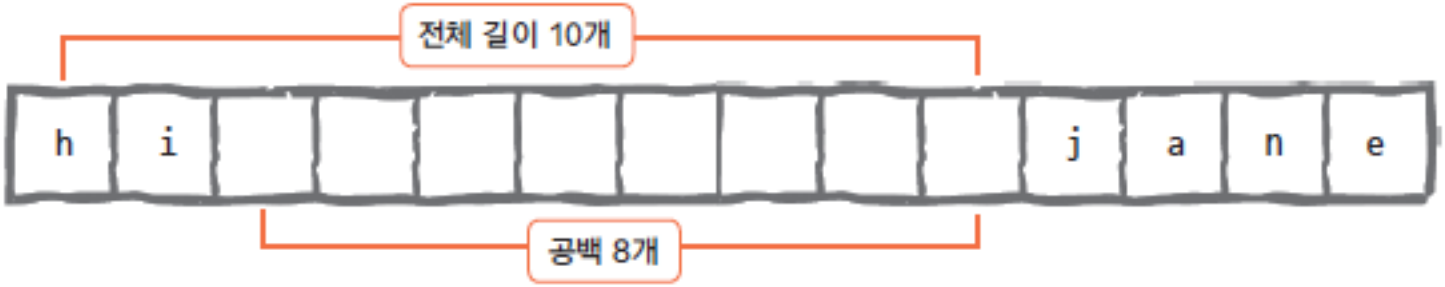
```
>>> number = 10
>>> day = "three"
>>> "I ate %d apples. so I was sick for %s days." % (number, day)
'I ate 10 apples. so I was sick for three days.'
```

문자열 포매팅

- 포멧 코드 정렬
  - 포멧 코드에 숫자를 이용하여 문자열을 정렬할 수 있음

```
>>> "%10s" % "hi"
'        hi'
```


```
>>> "%-10sjane." % 'hi'
'hi        jane.'
```



문자열 포매팅

- 포멧 코드 정렬
  - 실수의 소수점 자릿수를 설정할 수 있음
  - %0.4 혹은 %.4로 사용 가능

```
>>> "%0.4f" % 3.42134234
'3.4213'
```

format 함수를 사용한 포매팅

- 숫자 대입

```
>>> "I eat {0} apples".format(3)
'I eat 3 apples'
```

- 문자열 대입

```
>>> "I eat {0} apples".format("five")
'I eat five apples'
```

- 변수로 대입

```
>>> number = 3
>>> "I eat {0} apples".format(number)
'I eat 3 apples'
```

format 함수를 사용한 포매팅

- 이름으로 대입
  - {number}, {day}가 format 함수의 입력 값인 number=10, day=3 값으로 각각 바뀜

```
>>> "I ate {number} apples. so I was sick for {day} days.".format(number=10, day=3)
'I ate 10 apples. so I was sick for 3 days.'
```

- 왼쪽 정렬
  - :<10 표현식을 사용하면 문자열을 왼쪽으로 정렬하고 문자열의 총 자릿수를 10으로 맞춤

```
>>> "{0:<10}".format("hi")
'hi        '
```

- 오른쪽 정렬

```
>>> "{0:>10}".format("hi")
'        hi'
```

format 함수를 사용한 포매팅

- 가운데 정렬
  - :^를 사용하면 가운데 정렬이 가능

```
>>> "{0:^10}".format("hi")
'    hi    '
```

- 공백 채우기
  - 채워 넣을 문자 값은 정렬 문자 <, >, ^ 바로 앞에 넣어야 한다.

```
>>> "{0:=^10}".format("hi")
'====hi===='
>>> "{0:!<10}".format("hi")
'hi!!!!!!!!'
```

- 소수점 포현

```
>>> y = 3.42134234
>>> "{0:0.4f}".format(y)
'3.4213'
```

format 함수를 사용한 포매팅

- 가운데 정렬
  - :^를 사용하면 가운데 정렬이 가능

```
>>> "{0:^10}".format("hi")
'    hi    '
```

- 공백 채우기
  - 채워 넣을 문자 값은 정렬 문자 <, >, ^ 바로 앞에 넣어야 한다.

```
>>> "{0:=^10}".format("hi")
'====hi===='
>>> "{0:!<10}".format("hi")
'hi!!!!!!!!'
```

- 소수점 표현

```
>>> y = 3.42134234
>>> "{0:0.4f}".format(y)
'3.4213'
```

f-string 포매팅

- 문자열 앞에 접두사 'f'를 붙이면 f-string 포매팅이 가능
- (주의) 파이썬 3.6버전부터 지원

```
>>> name = '홍길동'
>>> age = 30
>>> f'나의 이름은 {name}입니다. 나이는 {age}입니다.'
'나의 이름은 홍길동입니다. 나이는 30입니다.'
```

- 정렬

```
>>> f'{"hi":<10}'  # 왼쪽 정렬
'hi        '
>>> f'{"hi":>10}'  # 오른쪽 정렬
'        hi'
>>> f'{"hi":^10}'  # 가운데 정렬
'    hi    '
```

f-string 포매팅

• 공백 채우기

```
>>> f'{"hi":=^10}'  # 가운데 정렬하고 '=' 문자로 공백 채우기
'====hi===='
>>> f'{"hi":!<10}'  # 왼쪽 정렬하고 '!' 문자로 공백 채우기
'hi!!!!!!!!'
```

• 소수점

```
>>> y = 3.42134234
>>> f'{y:0.4f}'  # 소수점 4자리까지만 표현
'3.4213'
>>> f'{y:10.4f}'  # 소수점 4자리까지 표현하고 총 자리수를 10으로 맞춤
'    3.4213'
```

문자열 내장 함수

- count()
  - 문자열에 포함된 문자 개수를 리턴함

```
>>> a = "hobby"
>>> a.count('b')
2
```

- find()
  - 문자열 중 특정 문자가 처음으로 나온 위치를 반환
  - 만약 찾는 문자나 문자열이 존재하지 않는다면 -1을 반환

```
>>> a = "Python is the best choice"
>>> a.find('b')
14
>>> a.find('k')
-1
```

문자열 내장 함수

- index()
  - 문자열 중 특정 문자가 처음으로 나온 위치를 반환
  - 만약 찾는 문자나 문자열이 존재하지 않는다면 오류를 발생

```
>>> a = "Life is too short"
>>> a.index('t')
8
>>> a.index('k')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: substring not found
```

- join()
  - abcd 문자열의 각각의 문자 사이에 ','를 삽입

```
>>> ",".join('abcd')
'a,b,c,d'
```

문자열 내장 함수

- upper()
  - 소문자를 대문자로 변경

```
>>> a = "hi"
>>> a.upper()
'HI'
```

- lower()
  - 대문자를 소문자로 변경

```
>>> a = "HI"
>>> a.lower()
'hi'
```

- lstrip()
  - 왼쪽 공백 지우기

```
>>> a = " hi "
>>> a.lstrip()
'hi '
```

문자열 내장 함수

- rstrip()
  - 오른쪽 공백 지우기

```
>>> a= " hi "
>>> a.rstrip()
' hi'
```

- strip()
  - 양쪽 공백 지우기

```
>>> a = " hi "
>>> a.strip()
'hi'
```

- replace()
  - 문자열 바꾸기

```
>>> a = "Life is too short"
>>> a.replace("Life", "Your leg")
'Your leg is too short'
```

문자열 내장 함수

- split()
  - 문자열 나누기

```
>>> a = "Life is too short"
>>> a.split()
['Life', 'is', 'too', 'short']
>>> b = "a:b:c:d"
>>> b.split(':')
['a', 'b', 'c', 'd']
```

형변환

- 형변환(casting)은 데이터를 다른 자료형으로 변경하는 것을 말함

```
>>> float(3)    #실수형으로 바꿈
3.0
>>> int(3.0)    #정수형으로 바꿈
3
>>> str(3)  #문자열로 바꿈
'3'
```

```
>>> hex(12) #16진수로 바꿈
'0xa'
>>> oct(10) #8진수로 바꿈
'0o12'
>>> bin(10) #2진수로 바꿈
'0b1010'
```

Q&A