IPA 주관 인공지능센터 기본(fundamental) 과정

- GitHub link: here
- E-Mail: windkyle7@gmail.com

구문(Statement)과 식(Expression)

구문(statement) = 문

구문 또는 신택스(syntax)란 프로그래밍 언어에서 프로그램의 모습, 형태, 구조가 어떻게 보이는지에 대해 정의하는 것이며, 구문은 정해진 문법을 이용한다.

- 예약어(reserved word, keyword)와 표현식을 결합한 패턴
- 컴퓨터가 수행해야 하는 하나의 단일 작업(instruction)을 명시.
- 할당 (대입, assigning statement)
  - python에서는 보통 '바인딩(binding)'이라는 표현을 씀, 어떤 값에 이름을 붙이는 작업.
- 선언(정의, declaration)
  - 재사용이 가능한 독립적인 단위를 정의. 별도의 선언 문법과 그 내용을 기술하는 블럭 혹은 블럭들로 구성.
    - Ex) python에서는 함수나 클래스를 정의
  - 블럭
    - 여러 구문이 순서대로 나열된 덩어리
    - 블럭은 여러 줄의 구문으로 구성되며, 블럭 내에서 구문은 위에서 아래로 쓰여진 순서대로 실행.
    - 블럭은 분기문에서 조건에 따라 수행되어야 할 작업이나, 반복문에서 반복적으로 수행해야 하는 일련의 작업을 나타낼 때 사용하며, 클래스나 함수를 정의할 때에도 쓰임.
- 조건(분기) : 조건에 따라 수행할 작업을 나눌 때 사용.
  - Ex) if 문
- 반복문 : 특정한 작업을 반복수행할때 사용.
  - Ex) for 문 및 while 문
- 예외처리

표현식(expression) = 평가식 = 식

- 값
- 값들과 연산자를 함께 사용해서 표현한 것
- 이후 '평가'되면서 하나의 특정한 결과값으로 축약
  - 수
    - 1 + 1
      - 1 + 1 이라는 표현식은 평가될 때 2라는 값으로 계산되어 축약
      - 0과 같이 값 리터럴로 값을 표현해놓은 것
    - 문자열
      - 'hello world'
      - "hello" + ", world"
    - 함수
      - lambda
- 궁극적으로 '평가'되며, 평가된다는 것은 결국 하나의 값으로 수렴한다는 의미
  - *python에서는 기본적으로 left-to-right로 평가*

이번 장에서는 값을 할당하는 여러 가지 방식에 대해 알아보고자 한다.

In [1]:
```
a = 1
a = b = c = a
a += 1
```

In [2]:
```
a, b = 1, 2,
```

In [3]:
```
a
```

Out[3]: 1

In [4]:
```
b
```

Out[4]: 2

In [5]:
```
c
```

Out[5]: 1

In [6]:
```
a, *b, c = 1, 2, 3, 4, 5
```

In [7]:
```
a
```

Out[7]: 1

In [8]:
```
b
```

Out[8]: [2, 3, 4]

In [9]:
```
a, b = {1, 2}
```

In [10]:
```
a
```

Out[10]:
```
1
```

In [11]:
```
{3, 1, 2, 3, 1}
```

Out[11]:
```
{1, 2, 3}
```

밑의 코드의 경우, `*`는 나머지 의미로 사용한다.

In [12]:
```
*a, b, c = "안녕하세요"
```

In [13]:
```
a
```

Out[13]:
```
['안', '녕', '하']
```

In [14]:
```
b
```

Out[14]:
```
'세'
```

In [15]:
```
c
```

Out[15]:
```
'요'
```

In [16]:
```
a = 5
if a > 0:
    print('Hi')
    print('안녕')
```

```
Hi
안녕
```

In [17]:
```
a = -1
if a > 0:
    print('Hi')
    print('안녕')
else:
    print('bye')
```

```
bye
```

In [18]:
```
a = 1000
if a > 0:
    print('Hi')
    print('안녕')
elif a > 5:
    print('bye')
else:
    print('잘가')
```

```
Hi
안녕
```

In [19]:
```
if 2:
    print('True')
```

```
True
```

In [20]:
```
if [2]:
    print('True')
```

```
True
```

In [21]:
```
if []:
    print('True')
else:
    print('False')
```

```
False
```

In [22]:
```
None == False
```

Out[22]:
```
False
```

None은 type 자체가 None이다.

In [23]:
```
type(None)
```

Out[23]:
```
NoneType
```

논리식은 아래와 같이 사용할 수도 있다.

In [24]:
```
a = 3
if 1 < a < 5:
    print('End')
else:
    print('Game')
```

```
End
```

아래의 경우, and 연산 시 앞(`1`)이 True면 뒤(`2`)의 값도 True인지 확인한다.

In [25]:
```
1 and 2
```

Out[25]: 2

In [26]:
```
3 and 2
```

Out[26]: 2

앞(`[]`)이 False면 더 이상 뒤의 값을 확인할 필요가 없으므로 False를 반환한다.

In [27]:
```
[] and 2
```

Out[27]: []

아래의 경우, `or` 연산 시 앞의 값이 False면 뒤의 값이 True인지 확인하여 (`2`)는 True이므로 `2`를 반환한다.

In [28]:
```
[] or 1
```

Out[28]: 1

In [29]:
```
1 or 1 / 0
```

Out[29]: 1

In [30]:
```
1 and 1 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-30-d1d23cf0694e> in <module>
----> 1 1 and 1 / 0

ZeroDivisionError: division by zero
```

In [31]:
```
a = 10
if a > 0:
    if a < 5:
        print('Hi')
    else:
        print('Bye')
```

```
Bye
```

In [32]:
```
a = 3
b = 3 if a > 0 else 8
```

In [33]:
```
a
```

Out[33]: 3

In [34]:
```
b
```

Out[34]: 3

In [35]:
```
a, b = range(2)
```

In [36]:
```
a
```

Out[36]: 0

In [37]:
```
b
```

Out[37]: 1

반복문: for

`for`는 다음과 같이 사용한다.

```
for 요소(식별자) in 이터러블(Iterable)_객체:
    ...
```

In [38]:
```
for i in [1, 2, 3]:
    print(i)
```

```
1
2
3
```

In [39]:
```
for i in {1, 2, 3}:
    print(i)
```

```
1
2
3
```

In [40]:
```
for i in {5, 2, 1}:
    print(i)
```

```
1
2
5
```

-

In [41]:
```
# Tuple
for i in 1, 2, 3:
    print(i)
```

```
1
2
3
```

In [42]:
```
for i in '엔드게임 보고싶어..':
    print(i)
```

```
엔
드
게
임

보
고
싶
어
.
.
```

In [43]:
```
x = 1, 2, 3, 4, 5
x
```

Out[43]:
```
(1, 2, 3, 4, 5)
```

반복을 완료했음에도 불구하고 식별자 `i`가 정의되어 있다는 것을 확인할 수 있다.

In [44]:
```
i
```

Out[44]:
```
'.'
```

아래는 반복문을 이용하여 구구단을 출력하는 코드이다.

In [45]:
```
for i in range(1, 10):
    print(2, '*', i, '=', 2 * i)
```

```
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
```

In [46]:
```
for i in range(2, 10):
    for j in range(1, 10):
        print(i, '*', j, '=', i * j)
    print('---------------')
```

```
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
---------------
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
---------------
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24
4 * 7 = 28
4 * 8 = 32
4 * 9 = 36
---------------
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
---------------
6 * 1 = 6
```

```
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
6 * 5 = 30
6 * 6 = 36
6 * 7 = 42
6 * 8 = 48
6 * 9 = 54
--------------
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
--------------
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
--------------
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
--------------
```

`for` 구문 다음에 오는 `else` 구문은 반복이 완료되면 실행되는 구문이다.

In [47]:
```
for j in range(2, 10):
    print(2, '*', j, '=', 2 * j)
else:
    print('a')
#
```

```
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
a
```

In [48]:
```
for i in range(1, 10):
    if i % 2 == 0:
        print(2, '*', i, '=', 2 * i)
```

```
2 * 2 = 4
2 * 4 = 8
2 * 6 = 12
2 * 8 = 16
```

In [49]:
```
for i in range(1, 10):
    if i % 2 != 0:
        print(2, '*', i, '=', 2 * i)
```

```
2 * 1 = 2
2 * 3 = 6
2 * 5 = 10
2 * 7 = 14
2 * 9 = 18
```

In [50]:
```
for j in {'a': 1, 'b': 2, 'c': 3}:
    print(j)
```

```
a
b
c
```

In [51]:
```
for j in {'a': 1, 'b': 2, 'c': 3}.keys():
    print(j)
```

```
a
b
c
```

In [52]:
```
for j in {'a': 1, 'b': 2, 'c': 3}.values():
    print(j)
```

```
1
2
3
```

In [53]:
```
for j in {'a': 1, 'b': 2, 'c': 3}.items():
    print(j)
```

```
('a', 1)
('b', 2)
('c', 3)
```

In [54]:
```
for i, j in {'a': 1, 'b': 2, 'c': 3}.items():
    print(i, j)
```

```
a 1
b 2
c 3
```

Accumulation Pattern

In [55]:
```
temp = 0
for i in range(1, 11):
    temp += i
else:
    print(temp)
```

```
55
```

In [56]:
```
[1, 2, 3, 4].index(3)
```

Out[56]:
```
2
```

반복문: while

반복문 `while` 은 다음과 같이 사용한다.

```
while 식:
    ...
```

종료 조건을 넣어주지 않으면 `무한 루프` 에 빠질 수 있으니 유의해야 한다.

In [57]:
```
a = 0
while a < 10:
    a += 1
    print(a)
```

```
1
2
3
4
5
6
7
8
9
10
```

반복문을 빠져나갈 조건을 어디에 넣어주는가가 중요하다.

In [58]:
```
a = 0
while a < 10:
    print(a)
    a += 1
```

```
0
1
2
3
4
5
6
7
8
9
```

In [59]:
```
a = 0
while a < 10:
    print(a)
    a += 1
else:
    print('Done')
```

```
0
1
2
3
4
5
6
7
8
9
Done
```

`continue` 는 반복문을 실행할 때 `continue` 다음 코드를 실행하지 않고 반복을 수행한다.

In [60]:
```python
# 무한 루프를 실행하는 코드이므로, 멈출 때는 i키를 2~3번 누른다.
a = 0
while a < 10:
    print(a)
    if a == 5:
        continue
    a += 1
else:
    print('Done')
```

```
0
1
2
3
4
5
5
5
5
5
5
5
5
5
5
5
5
5
5
5
5
5

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-60-53507cc6a4ef> in <module>
      2 a = 0
      3 while a < 10:
----> 4     print(a)
      5     if a == 5:
      6         continue

~/workspace/.venv/lib/python3.6/site-packages/ipykernel/iostream.py in write(self, string)
    398             is_child = (not self._is_master_process())
    399             # only touch the buffer in the IO thread to avoid races
--> 400             self.pub_thread.schedule(lambda : self._buffer.write(string))
    401             if is_child:
    402                 # newlines imply flush in subprocesses

~/workspace/.venv/lib/python3.6/site-packages/ipykernel/iostream.py in schedule(self, f)
    201             self._events.append(f)
    202             # wake event thread (message content is ignored)
--> 203             self._event_pipe.send(b'')
    204         else:
    205             f()

~/workspace/.venv/lib/python3.6/site-packages/zmq/sugar/socket.py in send(self, data, flags, copy, track, routing_id, group)
    393                                  copy_threshold=self.copy_threshold)
    394             data.group = group
--> 395         return super(Socket, self).send(data, flags=flags, copy=copy, track=track)
    396 
    397     def send_multipart(self, msg_parts, flags=0, copy=True, track=False, **kwargs):

zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.send()

zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.send()

zmq/backend/cython/socket.pyx in zmq.backend.cython.socket._send_copy()

~/workspace/.venv/lib/python3.6/site-packages/zmq/backend/cython/checkrc.pxd in zmq.backend.cython.checkrc._check_rc()

KeyboardInterrupt: 
```

In [61]:
```python
a = 0
while a < 10:
    print(a)
    a += 1
    if a == 5:
        continue
else:
    print('Done')
```

```
0
1
2
3
4
5
6
7
8
9
Done
```

In [62]:
```python
a = 0
while a > 10:
    print(a)
```

```
    a += 1
    if a == 5:
        continue
else:
    print('Done')
```

```
Done
```

In [63]:
```
temp = 0
for i in [1, 2, 3, 4, 1, 1]:
    if i == 1:
        temp += 1
else:
    print(temp)
```

```
3
```

`input` 함수는 사용자로부터 입력을 받은 뒤 문자열로 반환한다.

In [66]:
```
a = input()
```

```
엔드게임 보고싶다구..
```

In [67]:
```
a
```

Out[67]:
```
'엔드게임 보고싶다구..'
```

In [68]:
```
while True:
    a = input('5! = ')
    if a == '120':
        break
```

```
5! = 1
5! = 7
5! = 9
5! = 165
5! = 120
```

In [69]:
```
x = 3
for i in range(1, 10):
    print(x, '*', i, '=', x * i)
```

```
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
```

In [70]:
```
a, *b = 1, 2, 3, 4
```

In [71]:
```
a
```

Out[71]:
```
1
```

In [72]:
```
b
```

Out[72]:
```
[2, 3, 4]
```

In [73]:
```
a, b, c, d, *e  = '엔드게임 보고싶다..'
```

In [74]:
```
a
```

Out[74]:
```
'엔'
```

In [75]:
```
b
```

Out[75]:
```
'드'
```

In [76]:
```
e
```

Out[76]:
```
[' ', '보', '고', '싶', '다', '.', '.']
```

In [77]:
```
a, *b, c = '스포하지 말아줘.. ㅠㅠ'
```

In [78]:
```
b
```

Out[78]:
```
['포', '하', '지', ' ', '말', '아', '줘', '.', '.', ' ', 'ㅠ']
```

아래처럼 `*`는 단독으로 사용할 수 없다.

In [79]:
```
*e
```

```
  File "<ipython-input-79-e2f66073135d>", line 4
SyntaxError: can't use starred expression here
```

`,`를 붙여주면 실행이 되는데, 그 결과는 `tuple`이다.

In [80]:
```
*e,
```

Out[80]:
```
(' ', '보', '고', '싶', '다', '.', '.')
```

In [81]:
```
[a, b, c] = '파이썬'
```

In [82]:
```
a
```

Out[82]:
```
'파'
```

In [83]:
```
(a, b, c) = '파이썬'
```

In [84]:
```
b
```

Out[84]:
```
'이'
```

set은 순서가 없기 때문에 아래처럼 값을 할당할 수 없다.

In [85]:
```
{a, b, c} = '파이썬'
```

```
  File "<ipython-input-85-64ea4aa89ef1>", line 1
    {a, b, c} = '파이썬'
                ^
SyntaxError: can't assign to literal
```

In [86]:
```
c
```

Out[86]:
```
'썬'
```

while문을 사용하여 피보나치 구현하기

In [87]:
```
a, b = 0, 1
temp = 0
while temp < 8:
    temp += 1
    a, b = b, a + b
    print(b)
```

```
1
2
3
5
8
13
21
34
```

`for`문은 `while`문으로 대체하여 사용할 수 있지만, `while`문은 무한 루프를 할 수 있으므로 이 경우에는 대체 불가능하다.