Cpython 기초 구문

줄 구조

```
# -*- coding: <encoding-name> -*-          #<encoding-name>을 원하는 인코딩방식으로

aa=[1,7,3,4,5,6,7,8,9]                     #논리적인 줄 , 물리적인 줄
bb=(aa,)

a=[1,2,3,                                  #묵시적 줄 결합
   4,5,6,
   7,8,9]
b={123456
   ,78910}

print(bb#
    [0]#
    [1])                                   #명시적 줄 결합

if a[1]==1:                                #들여쓰기는 탭 문자나 공백
    if a[2]==2:                            #모두 허용되고 혼용도 가능
            a[3]=3                         #탭 문자는 공백으로 치환됨
                                           #들여쓰기에 따른 레벨은 스택의 구조로 측정
```

식별자

```
identifier   ::=  xid_start xid_continue*
id_start     ::=  <all characters in general categories Lu, Ll, Lt, Lm, Lo, Nl, the underscore, and characters with the Other_ID_Start property>
id_continue  ::=  <all characters in id_start, plus characters in the categories Mn, Mc, Nd, Pc and others with the Other_ID_Continue property>
xid_start    ::=  <all characters in id_start whose NFKC normalization is in "id_start xid_continue*">
xid_continue ::=  <all characters in id_continue whose NFKC normalization is in "id_continue*">
```

위에서 언급한 유니코드 카테고리 코드들의 의미는 이렇다:

- *Lu* - uppercase letters
- *Ll* - lowercase letters
- *Lt* - titlecase letters
- *Lm* - modifier letters
- *Lo* - other letters
- *Nl* - letter numbers
- *Mn* - nonspacing marks
- *Mc* - spacing combining marks
- *Nd* - decimal numbers
- *Pc* - connector punctuations
- *Other_ID_Start* - 하위 호환성 지원을 위해 PropList.txt 에서 명시적으로 나열된 문자들
- *Other_ID_Continue* - 마찬가지

키워드

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

```
def createGenerator():
    mylist = range(5)
    for j in mylist:
        yield j+2

generator = createGenerator()  #자바의 이터레이터와 같은 역할
                               #순환횟수가 많을 시 제너레이터 사용하면 메모리 절약가능
for i in generator:            #yield는 제네레이터 생성시 리턴할 값을 정의
    print(i)
```

```
2
3
4
5
6
Press any key to continue . . .
```

문자열과 바이트열 리터럴

```
stringliteral   ::=  [stringprefix](shortstring | longstring)
stringprefix    ::=  "r" | "u" | "R" | "U" | "f" | "F"
                     | "fr" | "Fr" | "fR" | "FR" | "rf" | "rF" | "Rf" | "RF"
shortstring     ::=  "'" shortstringitem* "'" | '"' shortstringitem* '"'
longstring      ::=  "'''" longstringitem* "'''" | '"""' longstringitem* '"""'
shortstringitem ::=  shortstringchar | stringescapeseq
longstringitem  ::=  longstringchar | stringescapeseq
shortstringchar ::=  <any source character except "₩" or newline or the quote>
longstringchar  ::=  <any source character except "₩">
stringescapeseq ::=  "₩" <any source character>
```

```
bytesliteral   ::=  bytesprefix(shortbytes | longbytes)
bytesprefix    ::=  "b" | "B" | "br" | "Br" | "bR" | "BR" | "rb" | "rB" | "Rb"
shortbytes     ::=  "'" shortbytesitem* "'" | '"' shortbytesitem* '"'
longbytes      ::=  "'''" longbytesitem* "'''" | '"""' longbytesitem* '"""'
shortbytesitem ::=  shortbyteschar | bytesescapeseq
longbytesitem  ::=  longbyteschar | bytesescapeseq
shortbyteschar ::=  <any ASCII character except "₩" or newline or the quote>
longbyteschar  ::=  <any ASCII character except "₩">
bytesescapeseq ::=  "₩" <any ASCII character>
```

```
str='good한글₩n'                    #일반 문자열
str2=r'good한글₩n'                  #raw string은 이스케이프 시퀀스를 무시하고 출력
print(str)
print(str2)

bstr=b'hello₩naa'                   #바이트 문자열
print(bstr)                         #'₩' 'n'을 따로 읽음
bstr2=br'hello₩naa'
print(bstr2)

print('₩n'+str+str2)
print(str*3)

name='추교준'
namestr=f'My name is {name}.'
print(namestr)
```

```
good한글

good한글₩n
b'hello₩naa'
b'hello₩₩naa'

good한글
good한글₩n
good한글
good한글
good한글

My name is 추교준.
Press any key to continue . . .
```

정수 리터럴

```
integer      ::=  decinteger | bininteger | octinteger | hexinteger
decinteger   ::=  nonzerodigit (["_"] digit)* | "0"+ (["_"] "0")*
bininteger   ::=  "0" ("b" | "B") (["_"] bindigit)+
octinteger   ::=  "0" ("o" | "0") (["_"] octdigit)+
hexinteger   ::=  "0" ("x" | "X") (["_"] hexdigit)+
nonzerodigit ::=  "1"..."9"
digit        ::=  "0"..."9"
bindigit     ::=  "0" | "1"
octdigit     ::=  "0"..."7"
hexdigit     ::=  digit | "a"..."f" | "A"..."F"
```

실수 리터럴

```
floatnumber   ::=  pointfloat | exponentfloat
pointfloat    ::=  [digitpart] fraction | digitpart "."
exponentfloat ::=  (digitpart | pointfloat) exponent
digitpart     ::=  digit (["_"] digit)*
fraction      ::=  "." digitpart
exponent      ::=  ("e" | "E") ["+" | "-"] digitpart
```

허수 리터럴

```
imagnumber ::=  (floatnumber | digitpart) ("j" | "J")
```

```
z=12_3                          #숫자 사이에 언더바 허용
print(z)
z=0xff                          #0+(b|o|x)+digit
print(z)
z=3.14e-10                      #digit|float + e + (+ | -) digit
print(z)
z=3j                            #digit|float + j
print(z*z)
```

```
123
255
3.14e-10
(-9+0j)
Press any key to continue . . .
```

연산자 구분자

```
+       -       *       **      /       //      %       @
<<      >>      &       |       ^       ~
<       >       <=      >=      ==      !=
```

```
(       )       [       ]       {       }
,       :       .       ;       @       =       ->
+=      -=      *=      /=      //=     %=      @=
&=      |=      ^=      >>=     <<=     **=
```