[데이터타입 (Data Types, 자료형) 개요]

[변수]는 아래와 같은 데이터타입(형)(자료형) 속성을 갖는다

- 수치 자료형: int (정수), float (실수), complex (복소수)

- 부울 자료형: bool (True / False)

- 컨테이너(군집 자료형) : str "문자열" , list [리스트], tuple (튜플), set {집합}, dict {사전}

[문자열] : String

문자열이란? : 문자의 나열

- 한 글자로 된 문자열 : "a", "가", "123"
- 단어로 된 문자열 : "boy", "소년"
- 문장으로 된 문자열 : "It rains.", "비가 옵니다."

문자열 변수 : 문자열 자료형(str)을 갖는 변수

In [ ]:
```
# 변수의 데이터 타입 (자료형) 비교
a = "Python is simple."
b = "123"
c = 123
d = 123.0
e = True
type(a),type(b),type(c),type(d),type(e) # 변수 데이터타입 확인
```

문자열를 출력하기

In [ ]:
```
print("Hello World")
```

In [ ]:
```
print("한글도 쓸 수 있어요.")
```

문자열을 만드는 다양한 방법

1) 큰따옴표로 양쪽 둘러싸기

In [ ]:
```
print ("Hello World")
```

2) 작은따옴표로 양쪽 둘러싸기

In [ ]:
```
print('Python is fun')
```

3) 큰따옴표 3개를 연속으로 써서 양쪽 둘러싸기

In [ ]:
```
print("""Life is too short, You need python""")
```

4) 작은따옴표 3개를 연속으로 써서 양쪽 둘러싸기

In [ ]:
```
print('''Life is too short, You need python''')
```

단순함이 자랑인 파이썬 인데, 왜 문자열 만드는 여러가지 방법이 있는가?

문자열 안에 작은따옴표나 큰따옴표를 포함시키고 싶을 때

1) 문자열에 작은따옴표 (') 포함시키기

- Python's favorite food is perl

In [ ]:
```
# print('Python's favorite food is perl') # 에러
```

In [ ]:
```
print("Python's favorite food is perl")
```

2) 문자열에 큰따옴표 (") 포함시키기

- "Python is very easy." he says.

In [ ]:
```
print('"Python is very easy." he says.')
```

(다른방법) : \ 백슬래쉬 (한글자판 원화표시) 이용해서 작은따옴표(')와 큰따옴표(")를 문자열에 포함시키기

In [ ]:
```
print('Python\'s favorite food is perl')
print("\"Python is very easy.\" he says.")
```

여러 줄인 문자열을 변수에 대입하고 싶을 때

- 문자열이 항상 한 줄짜리만 있는 것은 아니다.
- 다음과 같이 여러 줄의 문자열을 변수에 대입하려면 어떻게 처리해야 할까?

한 줄 쓰고
그 다음줄 쓴다.

In [ ]:
```
# 에러 : 여러줄로 출력 하고 싶을 때
# print("한 줄 쓰고
# 그 다음 줄을 쓴다.")
```

In [ ]:
```
print ("한 줄 쓰고")
print ("그 다음줄 쓴다.")
# 다른 방법은 없을까 ?
```

1) 연속된 작은따옴표 3개(''') 또는 큰따옴표 3개(""") 이용

- 위 1번의 단점을 극복하기 위해 작은따옴표 3개(''') 또는 큰따옴표 3개(""")를 이용한다.

In [ ]:
```
print ('''' Life is too short
 You need python''')
```

In [ ]:
```
print (""" 안녕 !!
 파이썬""")
```

In [ ]:
```
multiline = """
 안녕하세요
 반갑습니다.
 파이썬 입니다.
"""

print(multiline)
```

2) 줄을 바꾸기 위한 이스케이프 코드 삽입하기

- `print` 명령은 한 번 호출할 때마다 한 줄씩 출력한다.
- 만약 `print` 명령을 한 번만 쓰면서 여러 줄에 걸쳐 출력을 하고 싶으면 문자열에 "다음 줄 넘기기 (line feed) 기호"인 `\n` 를 넣어야 한다.

In [ ]:
```
print("한 줄 쓰고\n그 다음 줄을 쓴다.")
```

In [ ]:
```
print("Life is too short\nYou need python")
```

(참고) 줄을 바꾸지 않고 이어서 출력하기

반대로 `print` 명령을 여러번 쓰면서 줄은 바꾸지 않고 싶다면 다음과 같이 `print` 명령에 `end=""` 이라는 인수를 추가한다.

In [ ]:
```
print("한 줄 쓰고 ", end="")
print("이어서 쓴다.")
```

(참고) : 활용빈도 높은 이스케이프 코드

- 줄바꿈 \n
- 수평탭 \t
- 문자 \ \
- 홀따움표 \'
- 쌍타움표 \"

In [ ]:
```
 print("안녕하세요\n반갑습니다.")
```

In [ ]:
```
print("안녕하세요\t반갑습니다.")
```

- 한 줄 띄우기

`print` 명령은 한 번 호출할 때마다 한 줄씩 출력한다. 만약 `print` 명령을 한 번만 쓰면서 여러 줄에 걸쳐 출력을 하고 싶으면 문자열에 "다음 줄 넘기기(line feed) 기호"인 `\n`를 넣어야 한다.

In [ ]:
```
print("한 줄 쓰고\n그 다음 줄을 쓴다.")
```

[문자열 연산] : (+) (*)

- (+)연산은 두 문자열을 붙이고, (*) 연산은 문자열을 반복한다.

문자열 연결하기 (병합 : Concatenation) : +

- "문자열" + "문자열" +"문자열"

In [ ]:
```
12 + 34 , "12"+"34" # 숫자 와 문자열 연산 비교
```

In [ ]:
```
"12"+"34" # 결과값 46 ?
```

In [ ]:
```
print("내 이름은 " + "홍길동" + "입니다.")
```

In [ ]:
```
name = "홍길동" # 변수에는 숫자뿐만 아니라 문자열도 넣을 수 있다.
print("내 이름은 " + name + "입니다.")
```

문자열 반복하기 (Iteration) : *

- "문자열" * n (n은정수)
- a * 2라는 문장은 a를 두 번 반복하라는 뜻
- 즉, *는 문자열의 반복을 뜻하는 의미

In [ ]:
```
a = "python"
a * 2
```

In [ ]:
```
print("*" * 10)
```

In [ ]:
```
print("=" * 50)
print("제목 : Python Programing  +++ 작성자(아무개) +++ ")
print("=" * 50)
```

숫자를 문자열로 바꾸기 : str( )

- str("숫자형") --> 문자열
- 숫자를 문자열과 더하려면 `str` 명령을 써서 숫자를 문자열 자료형으로 바꾸어야 한다.

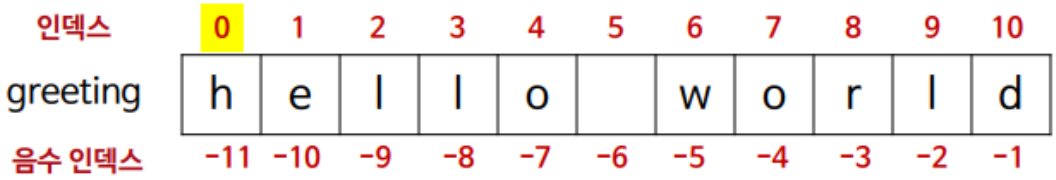
In [ ]:
```
# 문자열 과 숫자형 같이 출력하기
n = 25
# print("별표를 " + n + "번 출력합니다.") # 에러 : 문자열 + 숫자형 ???
print("*" * n)
print("별표를 " + str(n) + "번 출력합니다.")
print("*" * n)
```

[문자열 인덱싱, 슬라이싱]

- 인덱싱(Indexing) : 무엇인가를 "가리킨다": 문자열 순번 부여
- 슬라이싱(Slicing)은 무엇인가를 "잘라낸다" : 일부분 추출

문자열 인덱싱 : 변수명[i]

- 인덱싱 : a = "hello world"
- "파이썬은 0부터 숫자를 센다."
- a[0]:'L', a[1]:'i', a[2]:'f', a[3]:'e', a[4]:' ', ...
- a[번호]는 문자열 내 특정한 값을 뽑아내는 역할. 이러한 것을 **인덱싱** 이라고 한다.



In [ ]:
```
a = "Life is too short, You need Python"
```

- L은 첫 번째 자리를 뜻하는 숫자인 0, 바로 다음인 i는 1 이런 식으로 계속 번호를 붙인 것이다. 중간에 있는 short의 s는 12가 된다.

In [ ]:
```
a = "Life is too short, You need Python"
a[3]
```

In [ ]:
```
greeting = "hello world"
greeting[0], greeting[7], greeting[-1], greeting[-5]
```

In [ ]:
```
a = "Life is too short, You need Python"
a[0], a[12], a[-1]
```

In [ ]:
```
a[0],a[-0],a[-2],a[-5]
```

In [ ]:
```
a = "Life is too short, You need Python"
b = a[0] + a[1] + a[2] + a[3]
b
```

- 단순히 한 문자만을 뽑아내는 것이 아니라, 'Life' 또는 'You' 같은 단어들을 뽑아내는 방법은 없을까?

- 위의 방법처럼 단순하게 접근할 수도 있지만
- 더 좋은 방법 슬라이싱(Slicing) 이라는 기법이다.

문자열 슬라이싱: 변수명[n:m], "문자열"[i]

- 문자열 일부분을 추출

- 단순히 한 문자만을 뽑아내는 것이 아니라, 'Life' 또는 'You' 같은 단어들을 뽑아내는 방법은 없을까?

In [ ]:
```
"Hello"[1:4]
```

(주의) a[시작 번호:끝 번호]를 지정하면 끝 번호에 해당하는 것은 포함되지 않는다.

a[1:4] 1 <= a < 4 ("번호 앞을 자른다")

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

In [ ]:
```
a[0:3] # 0 <= a < 3
```

In [ ]:
```
a = "Life is too short, You need Python"
a[0:5], a[5:7], a[12:17]
```

In [ ]:
```
a = "안녕하세요 빅데이터 파이썬"
a[0:5], a[5:10], a[12:17]
```

In [ ]:
```
a = "Life is too short, You need Python"
a[19:]
```

In [ ]:
```
a[:17]
```

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

In [ ]:
```
a[19:-7]
```

- 슬라이싱으로 문자열 나누기

In [ ]:
```
a = "20010331Rainy"
date = a[:8]
weather = a[8:]
date,weather
```

In [ ]:
```
a = "20010331Rainy"
year = a[:4]
day = a[4:8]
weather = a[8:]
year, day, weather
```

(주의) "Pithon"이라는 문자열을 "Python"으로 바꾸려면?

In [ ]:
```
a = "Pithon"
# a[1] = 'y'
```

In [ ]:
```
# 문자 변경이 불가하여, 새롭게 조합
a = "Pithon"
a[:1] # 'P'
a[2:] # 'thon'
a[:1] + 'y' + a[2:]
```

[문자열 메소드(Method)] : Built-in String Methods

문자열 메소드 : 문자열.메소드( ) [변수/객체.메소드()]

- 문자열에만 적용할 수 있는 함수들
- dir(str)이라고 하면 문자열 메소드 목록을 볼 수 있음
- **문자열.메소드( )** 형태로 사용 (메소드는 객체에 어떤 일을 처리하도록 하는 코드)
- 파이썬은 기본적으로 문자열을 변경이 불가능(immutable)하기 때문에 직접 문자열을 수정하는 방식이 아닌 변경된 다른 문자열을 리턴하는 방식
- 문자열 메소드 종류 : find(), lower(), upper(), lstrip(), rstrip(), split(), splitlines(), replace(), join(), zfill()

- 문자열 메소드 : 문자열에만 적용할 수 있는 함수들

- `dir(str)` 이라고 하면 문자열 메소드 목록을 볼 수 있음

- 종류 판별 (True, False) : .isdigital(), .isalpha()

- isalpha함수: 문자열이 문자인지 아닌지를 True,False로 리턴
- isdigit함수: 문자열이 숫자인지 아닌지를 True,False로 리턴

In [ ]:
```
# .isdigit() : 사용예 --> input() 숫자판별
a = "12345"
a.isdigit()
```

In [ ]:
```
a = "-123"
a.isdigit()
```

In [ ]:
```
# .isalpha()
b = "hi"
b.isalpha()
```

- 문자열과 문자열 연결하기 : .join()

In [ ]:
```
"--".join("abcd")
```

In [ ]:
```
a = "---"
a.join("abcdef")
```

- 문자열과 문자열 분리 (리스트) Splitting strings : 변수/객체.split( )

- a.split( )처럼 괄호 안에 아무런 값도 넣어 주지 않으면 **공백(스페이스, 탭, 엔터등)**을 기준으로 문자열을 나누어 준다.
- a.split(',')처럼 괄호 안에 특정한 값이 있을 경우에는 괄호 안의 값을 구분자로 해서 문자열을 나누어 준다.

In [ ]:
```
"Life is too short".split()
```

In [ ]:
```
'11/19/2020'.split('/')
```

In [ ]:
```
# for CSV files (예:CSV화일)
a = "2011, 2012, 2013, 2014, 2016"
a.split(",") # for CSV files
```

In [ ]:
```
# for whitespace
a = "2011 2012 2013 2014 2016"
a.split(" ") # for CSV files
```

In [ ]:
```
# 23490.split() # 숫자 에러
```

In [ ]:
```
date = '11/19/2020'
date.split('/')
```

In [ ]:
```
a = "Life is too short"
a.split()
b = a.split() # List 출력

print(type(a), a)
print(type(b), b)
```

- 문자열지우기: .strip("지울문자") / 공백 지우기 : .strip()

In [ ]:
```
"하이, 안녕하세요".strip("하이, ")
```

In [ ]:
```
"  정답은 사과입니다.    ".strip()
```

In [ ]:
```
# 양쪽 공백지우기
a = "  정답은 사과입니다.    "
a.strip()
```

In [ ]:
```
# 왼쪽 공백지우기 (lstrip)
a = "      정답은 사과입니다.    "
a.lstrip()
```

In [ ]:
```
# 오른쪽 공백지우기 (lstrip)
a = "           정답은 사과입니다.    "
a.rstrip()
```

- 문자열 바꾸기: .replace (바뀌게 될 문자열, 바꿀 문자열)

In [ ]:
```
a = "Life is too short"
a.replace("Life", "Your leg")
```

- 문자열 대소문자 바꾸기 : .upper() , .lower()

In [ ]:
```
a = "python"
a.upper()
```

In [ ]:
```
b = "PYTHON"
b.lower()
```

In [ ]:
```
book = 'Alice in Wonderland'
book2 = book.upper( ) # 문자열 book에 upper( ) 메소드를 적용함.
print(book2) # book2는 book을 모두 대문자로 만든 문자열임.
print(book) # book은 그대로임.
```

- 문자 개수 세기: .count()

In [ ]:
```
a = "hobby"
a.count('b') # 문자열 중 문자 b의 개수를 반환한다
```

- 문자 위치찾기 : . find('x'), index('x')

- 해당 문자가 처음 나온 위치 반환
- find() : 없는경우 -1
- index() : 없는경우 오류 -> 없는경우 처리에 사용

In [ ]:
```
# find()
a = "Python is the best choice"
a.find('b') # 파이썬은 숫자를 0부터 세기 때문에 b의 위치는 15가아닌 14가 된다.
```

In [ ]:
```
# find() 없는경우 -1
a.find('k') # 만약 찾는 문자나 문자열이 존재하지 않는다면 -1을 반환한다.
```

In [ ]:
```
# index()
a = "Life is too short"
a.index('t')
```

In [ ]:
```
# index() 없는경우 - 오류
#a.index('k')
```

- in, not in 연산자

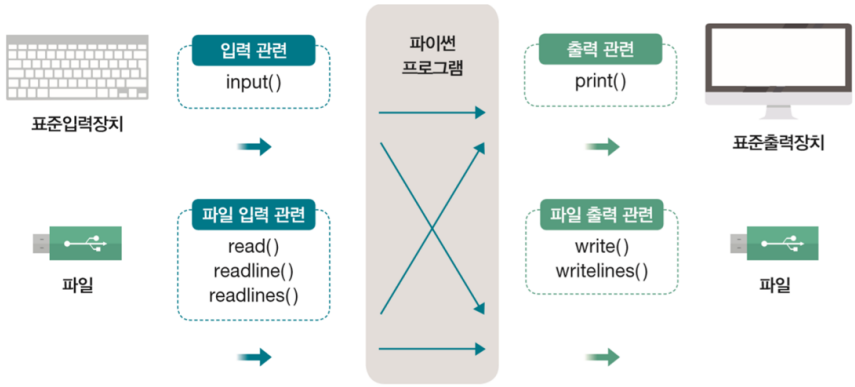
In [ ]:
```
subject = "programming"
'r' in subject, 'gram' in subject
```

In [ ]:
```
'abcd' not in subject
```

In [ ]:
```
# (기타) 문자열 길이
a = "Life is too short"
len(a)
```

[표준 입출력문]

- 입력과 출력 { 입력(키보드, 파일) >> 출력(모니터, 파일) }



표준 입력(키보드 입력) : input() 문사용

- input은 입력되는 모든 것을 "문자열"로 취급한다.

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

In [ ]:
```
number = input("숫자를 입력하세요: ")
print(number)
```

In [ ]:
```
name = input("여기는 무슨 대학교 입니까? ")
print("여기는", name, "입니다")
```

표준출력 (화면 출력) : print( ) 문 사용

(x, y, z) --> (x, y, z, sep = " ") : 문자띄어쓰기

(x y z) , (x+y+z)

(" %s " %" 문자열") , (" {} " . format( )) , (f" { } ")

- 값을 여러개 출력 (문자열 띄어쓰기는 콤마 사용)

- print(값1, 값2, 값3)
- print(변수1, 변수2, 변수3)

In [ ]:
```
# 문자열 띄어쓰기 콤마
print(1,2,3)
print("이름은", "김수원", "입니다.")

x = "김수원"
print("이름은", x, "입니다.") # 문자열, 변수, 문자열
```

- 문자열 붙여쓰기 (+ 또는 콤마없이)

In [ ]:
```
print("이름은" "김수원" "입니다.")
print("이름은" + "김수원" + "입니다.")
```

- 값사이에 문자넣기

- print(값1, 값2, sep='문자열')
- print(변수1, 변수2, sep='문자열')

In [ ]:
```
print(1,2,3, sep="          ")
print(1,2,3, sep=",    ")
print(1920, 1080, sep="x")
```

- 줄바꿈 활용하기

In [ ]:
```
print(1,2,3)
print(1,2,3, sep="\t") # 탭
print(1,2,3, sep="\n") # 줄바꿈
print("1\n2\n3")
```

- 한 줄에 결과값 출력하기 (end=" ")

In [ ]:
```
print(1, end='')     # end에 빈 문자열을 지정하면 다음 번 출력이 바로 뒤에 오게 됨
print(2, end='')
print(3)
```

문자열서식(format) : (" %s " %x), ("{ }" .format( )) , (f"{x} ")

- 파이썬에서는 복잡한 출력을 위한 문자열 형식화(string formatting)를 지원한다.
- 문자열을 형식화하는 방법에는 `%` 기호를 사용한 방식, `format` 메서드, `f"` 사용한 방식이 있다.

In [ ]:
```
"이름은 %s 입니다." %"김수원" # print ( ... )
```

In [ ]:
```
x = "이름은 %s 입니다."
print(x %"김수원")
```

In [ ]:
```
ss = "김수원"
```

In [ ]:
```
print("이름은 %s 입니다." %"김수원")
print("이름은 %s 입니다." % ss)
```

In [ ]:
```
print("이름은 {} 입니다.".format("김수원"))
print("이름은 {} 입니다.".format(ss))
```

In [ ]:
```
print(f"이름은 {ss} 입니다.")
```

1) 문자열 서식지정 (% 연산자)

- 데이터, 변수 등을 표시할 때 **"서식"(%d)** 을 적용
- 서식은 %로 시작
- "문자열" % 값

In [ ]:
```
a = 3
b = 3.0
c = "3"
print("%d and %f and %s" % (a,b,c))
```

In [ ]:
```
ss = "김수원"
print("이름은 %s 입니다." %"김수원")
print("이름은 %s 입니다." % ss)
```

In [ ]:
```
print("4 + 5 결과값은 %d 입니다" %9 )
```

In [ ]:
```
print("원주율의 값은 %f입니다." %3.141592 )
```

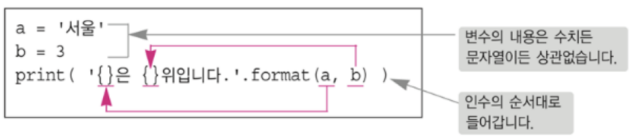
In [ ]:
```
print("%d 곱하기 %d은 %d이다." % (2, 3, 6)) # 문장중에 변경 가능 할 경우
```

In [ ]:
```
print("%s의 %s 과목 점수는 %d점이다." % ("철수", "수학", 100))
```

In [ ]:
```
a = "철수"
b = "수학"
c = 100
print ("%s의 %s 과목 점수는 %d점이다." % (a,b,c))
```

2) 문자열 서식지정 (str. format ( ) 함수 사용) : 파이썬 3 이후 사용

- 파이썬 format() 함수, {} 서식지정 방법
- .format 사용하면 출력 하는 순서를 지정가능
- 서식, 자리수, 정렬 방식 등을 지정
- `format` 메서드를 사용하여 문자열을 형식화할 수도 있다. 이 때는 % 기호로 시작하는 형식지정 문자열 대신 {} 기호를 사용한다. 또한 자료형을 표시할 필요가 없다.



In [ ]:
```
print("이름은 {}입니다.".format("김수원"))
```

In [ ]:
```
print("{}은 {}위 입니다 ".format("서울",3))
```

In [ ]:
```
a ="서울"
b = 3
print("{}은 {}위 입니다".format(a,b)) # 문장을 변수에 할당
```

In [ ]:
```
# 순서지정 가능
a ="한국"
b = 3
c = "세계 야구대회"
print("{2}에서 {0}은 {1}위 입니다".format(a,b,c)) # 순서 지정
```

In [ ]:
```
print("문자열 서식: {0:d} {1:5d} {2:05d} ".format(123,123,123))
```

3) 문자열 서식지정 (f-string) : 파이썬 3.6 이후

- String 앞에 f 를 붙이고 문자열 내에 { } 와 함께 변수를 바로 입력

```
a = '서울'
b = 3
print( f'{a}은 {b}위입니다.')
```

In [ ]:
```
a = "서울"
b = 3
print( f"{a}은 {b}위 입니다")
```

In [ ]:
```
a ="한국"
b = 3
c = "세계 야구대회"
print(f"{c}에서 {a}은 {b}위 입니다") # 순서 지정
```

---------- [end] ----------