

codingOn x posco

K-Digital Training 신재생에너지 활용 IoT 과정

Python 기초문법

오늘 수업은?

- 지금까지는 개발을 시작하기전 알아두어야할 내용에 대해서 공부 하였습니다.
- 이번 시간에는 본격적으로 파이썬을 배우면서 프로그래밍이 무엇 인지 알아보는 시간을 가지겠습니다.

학습목표

- 파이썬 프로그래밍 언어에 대해서 이해한다.
- 변수를 할당하고 사용하는 방법을 이해한다.
- 다양한 연산자를 활용할 수 있다.
- 파이썬의 자료형에 대해서 이해한다.
- 문자열을 활용하는 다양한 방법을 이해한다.

1. Python을 배우는 이유

















Python 장점

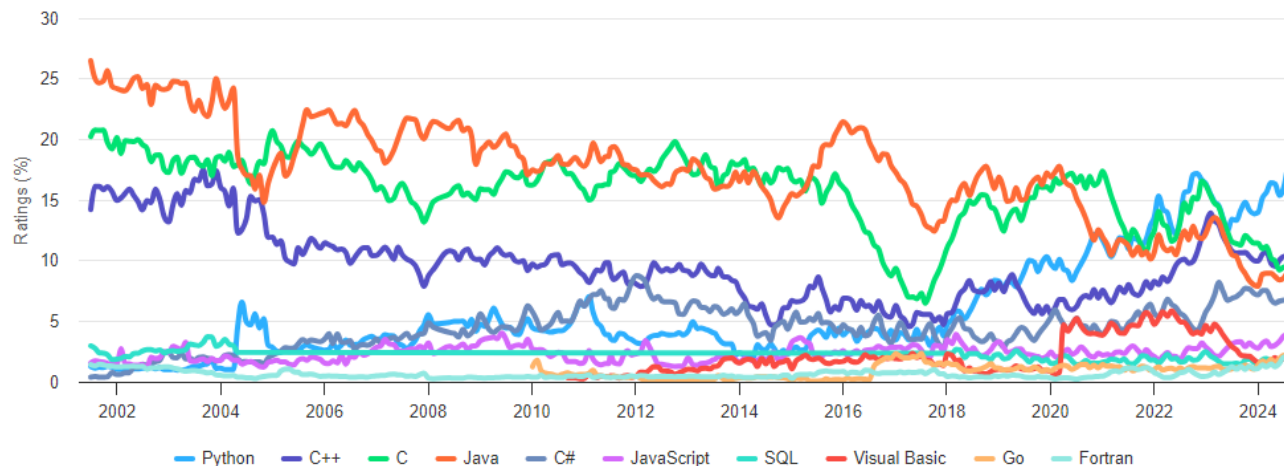
- 코드가 간결하고 문법이 쉽다.
- 다양한 분야 활용 가능
 - 예) 웹 개발, 해킹 도구, AI, 데이터 분석, 사물인터넷 등
- 많은 기능들을 갖고 있는 라이브러리가 풍부하다
- 개발 속도가 빠르다.

단점: 시스템 프로그램(주로 C언어), 모바일 앱 등에 아직 사용이 적다.

Python

- Python은 세계에서 가장 많이 사용되는 프로그래밍 언어 중 하나

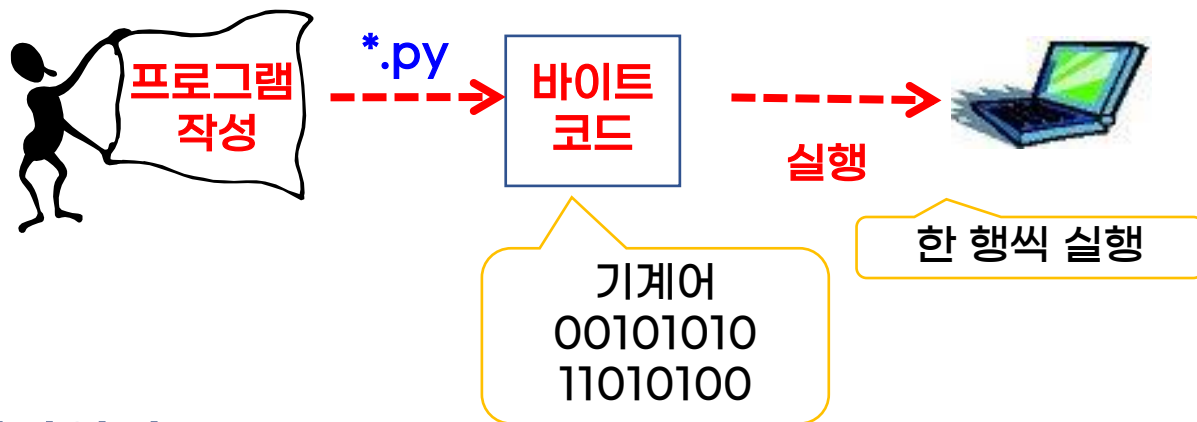
Aug 2024	Aug 2023	Change	Programming Language	Ratings	Change
1	1		 Python	18.04%	+4.71%
2	3	▲	 C++	10.04%	-0.59%
3	2	▼	 C	9.17%	-2.24%
4	4		 Java	9.16%	-1.16%
5	5		 C#	6.39%	-0.65%
6	6		 JavaScript	3.91%	+0.62%
7	8	▲	 SQL	2.21%	+0.68%
8	7	▼	 Visual Basic	2.18%	-0.45%
9	12	▲	 Go	2.03%	+0.87%
10	14	▲	 Fortran	1.79%	+0.75%
11	13	▲	 MATLAB	1.72%	+0.67%
12	23	▲	 Delphi/Object Pascal	1.63%	+0.83%
13	10	▼	 PHP	1.46%	+0.19%
14	19	▲	 Rust	1.28%	+0.39%
15	17	▲	 Ruby	1.28%	+0.37%
16	18	▲	 Swift	1.28%	+0.37%



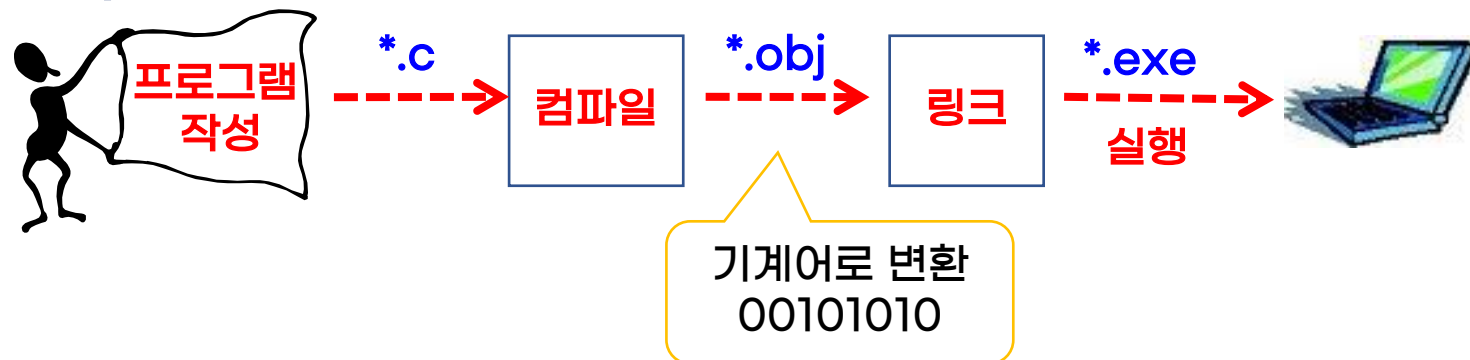
출처: TIOBE (티오베)

Python은 인터프리터 방식

➤ 인터프리터(Interpreter)



➤ 컴파일러 (Compiler)



Python 문서

자습서 : <https://docs.python.org/ko/3/tutorial/index.html>

라이브러리 : <https://docs.python.org/ko/3/library/index.html>

- 2. 파이썬 인터프리터 사용하기
 - 2.1. 인터프리터 호출
 - 2.1.1. 인수 전달
 - 2.1.2. 대화형 모드
 - 2.2. 통역사와 그 환경
 - 2.2.1. 소스 코드 인코딩
- 3. 파이썬에 대한 비공식적인 소개
 - 3.1. 파이썬을 계산기로 사용하기
 - 3.1.1. 숫자
 - 3.1.2. 텍스트
 - 3.1.3. 목록
 - 3.2. 프로그래밍을 향한 첫 걸음
- 4. 더 많은 제어 흐름 도구
 - 4.1. if 진술
 - 4.2. for 진술
 - 4.3. range() 기능

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5 # division always returns a floating
1.6
```

2. 프로그래밍 시작하기

시작하기

- 개발자가 하는 업무는 크게 보면 어떠한 문제를 해결하기 위한 일
 - 예) 배달을 전화로 하기 귀찮아 하는 사람들이 많아짐(문제점)
⇒ 배달의 민족, 요기요 등 간단하게 주문할 수 있는 앱이 생기게 됨(해결)
- 앞으로 여러분들이 코딩하는 것들은 어떤 기능을 어떻게 개발하여 문제점을 해결 할 것인가에 대한 내용들임
- 그래서 개발 잘 하려면 문제 해결 능력을 키워야 함
- 그리고 우리는 능력을 키우기 위한 기본기를 향상 시킬 것 임
- 앞으로 프로그래밍 언어를 배우면서 “왜 이런개념을 배워야하는지” 그리고 “배운 내용을 어떻게 사용해야하는지”에 대해서 공부할 예정

출력: print()

- 메시지 출력 기능을 담당하는 함수
- 여러 개 출력 시 쉼표 (,)로 구분 -> 무조건 띄워 쓰기가 하나 들어감
 - 띄워 쓰기 없애려면 sep 옵션 추가(sep는 단어사이 분리할 문자 넣는 기능)

```
print("Hello", "Python") # Hello Python
print("Hello", "World", sep="") # HelloWorld
```

- 문자열, 숫자 가능
- 괄호 안에 아무것도 넣지 않으면 줄 바꿈 역할
- 이어서 출력하고 싶다면, end 옵션을 추가하기. 줄바꿈을 하지 않음
 - end는 중간에 넣고 싶은 값을 넣는 기능

```
print("안녕하세요", end=" ")
print("코딩온입니다")
# 위 두개 실행 시 결과 "안녕하세요 코딩온입니다"
```

입력: input()

- 메시지 입력 기능을 담당하는 함수
- 사용자마다 다른 값을 입력 받아서 사용
- input("프롬프트 문자열")
- 주의) input으로 입력 받은 값은 문자열

```
singer = input("좋아하는 가수는?")  
print("좋아하는 가수는", singer, "입니다")
```

```
좋아하는 가수는?뉴진스  
좋아하는 가수는 뉴진스 입니다
```

주석

- 프로그램의 진행에 전혀 영향을 주지 않는 코드
- 프로그램을 설명하기 위해 사용
- # 사용 - 한 문장 처리
- 독스트링(docstrings) : 쌍따옴표 or 홀따옴표 3개 - 여러줄 문장
(독스트링에 대한 내용은 교안 후반부에 또 다시 나옵니다)

```
# 한 줄 주석시 사용, 코드뒤에서도 사용이 가능
x = 10 # 주석사용

"""
긴 문장을 주석처리할 경우
시작위치와 끝나는위치에
쌍따옴표를 3개씩 넣으면 됨
"""
...
홀따옴표도 가능
...
```

들여쓰기

- 파이썬에서는 코드의 구조를 명확하게 하고, 블록을 구분하기 위해서 들여쓰기를 문법적으로 강제하고 있음
- 조건문, 반복문, 함수, 클래스 등에서 사용됨
- 들여쓰기가 정확하지 않으면 오류가 발생

```
if age >= 18:  
    print("성인입니다.")  
else:  
    print("미성년자입니다.")
```

들여쓰기부분

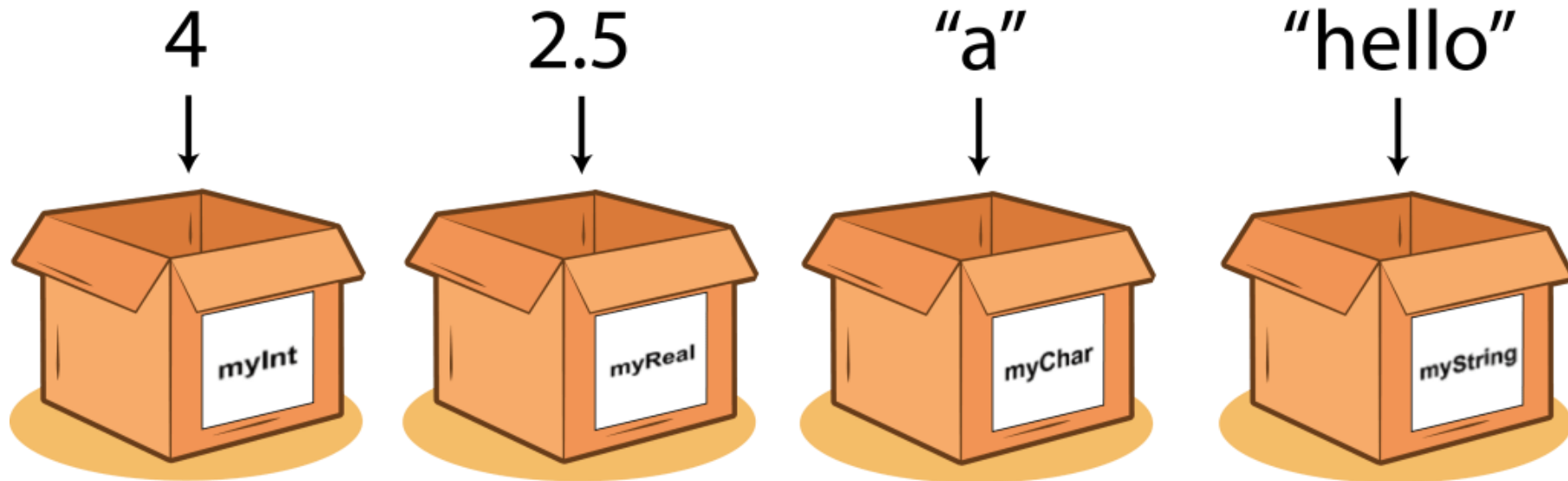
객체

- 파이썬에서 다루는 데이터는 모두 객체
- 앞으로 배울 숫자, 문자열, 리스트, 딕셔너리 등등 모든 것들
- 객체란 데이터와 그 데이터를 처리하는 매서드를 묶어서 하나로 만든 개념
- 특징
 - 가변성: 리스트, 딕셔너리, 셋과 같은 가변 객체는 값을 수정할 수 있음
 - 불변성: 숫자, 문자열, 튜플 같은 불변 객체는 값을 수정할 수 없고, 값을 변경하면 새로운 객체가 생성
- => 지금은! 처음보는 용어가 나온다고 당황하거나 어렵게 생각하지 마세요!
- => 앞으로 하나하나씩 배워나가면서 계속 사용할 단어입니다.

3. 변수

변수

- 변수 = variable, 변하는 값
- 변수는 데이터를 담는 빈 그릇!



변수 값 할당

- 변수이름 = 변수에 저장할 값
 - `x = 10` (정수형)
 - `y = 3.14` (실수형)
 - `z = "Hello"` (문자열)
- 변수에 값을 가지도록 하는것을 “**변수에 값을 할당한다**”라고 함
- 변수에 저장된 값은 언제든지 다른 값으로 재할당 할 수 있음
 - `x = 5`
 - `x = "Python"`
- 여러 개의 변수를 한번에 할당할 수도있음
 - `x, y, z = 10, 3.14, "Hello"`

왜 사용할까?

- 여러분들이 1부터 100까지 2를 곱하는 수식을 만들어서 출력한다고 가정
- 그렇다면 우측과 같이 곱셈 수식을 만들어야 함
- 그런데 갑자기 2가 아니고 3으로 변경된다면?
- 2의 값들을 전부 3으로 변경하면 되겠지만 1부터 100이 아닌 1부터 100000이었다면? 100000번을 수정해야하는 작업을 해야함
- 이때 필요한게 변수라는 것임

```
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
.
.
.
2 * 98 = 196
2 * 99 = 198
2 * 100 = 200
```

왜 사용할까?

- 앞선 예시에서 변수를 사용하게 된다면?

```
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
.
.
.
2 * 98 = 196
2 * 99 = 198
2 * 100 = 200
```

변수 사용 전

```
x = 2
x * 1 = 2
x * 2 = 4
x * 3 = 6
x * 4 = 8
x * 5 = 10
x * 6 = 12
x * 7 = 14
.
.
.
x * 98 = 196
x * 99 = 198
x * 100 = 200
```

변수 사용 후

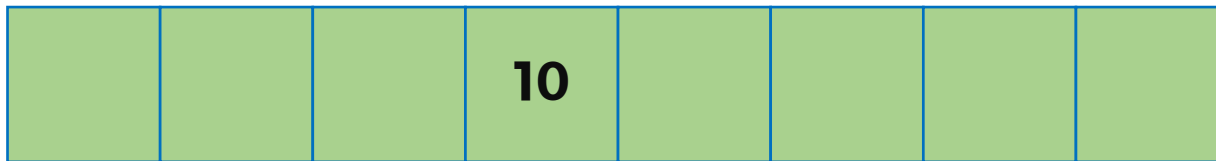
이제 x 변수에 값을 3으로 변경해 주면 끝

변수를 사용하게 된다면

- 변수를 사용하면 값을 저장하고 나중에 재사용할 수 있어 코드가 더 간결해짐
- 값을 변수에 저장하면 나중에 변경할 때 코드 전체를 수정할 필요 없이 변수만 변경하면 됨
- 계산 결과나 문자열을 변수에 저장하면 중복 작업을 줄일 수 있음
- 변수 이름을 통해 값의 의미를 명확히 알 수 있어, 코드를 더 직관적으로 작성할 수 있음
- 복잡한 수식을 변수로 분리하면 코드를 모듈화하여 쉽게 수정할 수 있음

변수를 사용할 수 있는 이유

- 변수에 값을 할당하게 되면 컴퓨터에 그 **값**을 메모리에 저장함
- 메모리란 컴퓨터가 데이터를 저장하고 처리하는 공간(RAM)
- $x = 10$ 일때
- 10은 메모리의 특정 위치에 저장되고 변수 x 는 그 위치를 참조(Reference)
- 변수가 값을 직접 저장하는 것이 아닌 변수는 값을 저장한 메모리 주소 위치를 가리키는 방식

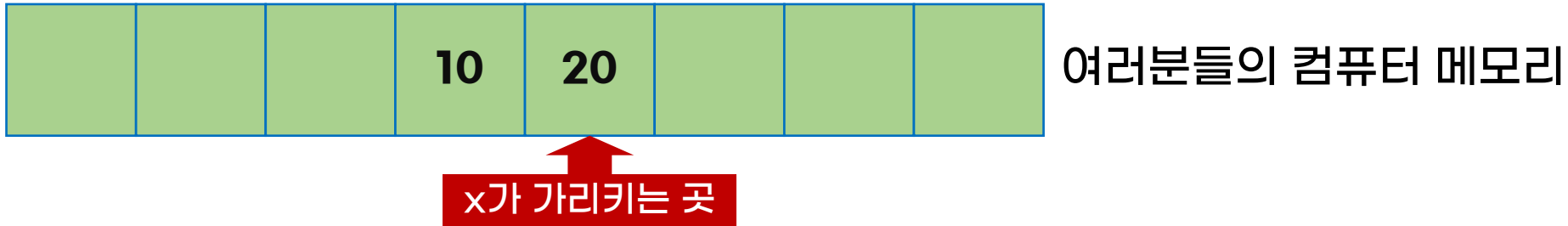


여러분들의 컴퓨터 메모리

x가 가리키는 곳

가비지 컬렉터

- 다시 재할당하여 $x = 20$ 이라고 가정



- 이제 10은 더 이상 참조되지 않으므로 파이썬의 **자동 메모리 관리인 가비지 컬렉터**가 메모리에서 자동으로 삭제
- 가비지 컬렉터는 파이썬, c#, java 등 프로그래밍 언어에서 메모리의 누수방지를 위한 관리 시스템임

변수의 두가지 타입(복습용)

- 타입1. 숫자, 문자열 등은 값을 변경한다하지 않고 새로운 값을 할당하면 새로운 메모리 위치를 참조 하는 것
- 타입2. 리스트, 딕셔너리 등은 값이 변경 될 수 있으며, 값이 변경되더라도 같은 메모리 위치를 계속 참조
- 메모리 주소확인 하는 방법
 - id(값)

• 예)

```
x = 10
print("before", id(x))
x = 20
print("after", id(x))
before 140710924270296
after 140710924270616
```

타입1

```
a = [1,2,3]
print('before', id(a))
a.append(4)
print('after', id(a))
before 2615740977472
after 2615740977472
```

타입2

상수

- 변수의 반대말로 **재할당하지 않는 값을 저장하는 것을 상수라고 함**
- 프로그래밍 중 값을 재할당하지 않지만 그 값을 사용할 때 상수를 이용함
- 예) 우편번호 13579를 저장해서 사용해야한다면
 - `POST_CODE = 13579`
- 예) 데이터베이스 이름 정보
 - `DB_NAME = "my_db"`

식별자

- 변수의 이름은 아무렇게 지을 수 없음
- 식별자는 프로그래밍 언어에서 **이름을 붙일 때** 사용하는 단어
- 변수뿐만 아니라 함수, 클래스, 모듈 등에서도 사용하는 단어
- 식별자 규칙은 관례상 사용하는 규칙이며 많은 개발자들이 이 관례를 따르고 있음

식별자 규칙

- 첫 문자는 알파벳 문자이거나 밑줄이어야 함. 숫자는 안됨
- 나머지 문자는 문자, 밑줄(_) 또는 숫자(0-9)여야 함
- 대소문자를 구분함
- 공백을 포함할 수 없고 특수문자도 사용할 수 없음
- 키워드(예약어)는 사용 불가
- 상수는 모두 대문자 형태
- snake_case : 단어사이를 밑줄(_)로 연결된 형태. 함수나 변수에 사용
 - 예) product_id = 1
- PascalCase : 단어사이의 첫글자를 대문자로 작성한 형태. 클래스에 사용
 - 예) class CarType:

hi apple (x) → 공백
hello* (x) → 특수문자
hello_ (o) → 특수문자이기는 하지만, _ 허용
hello23 (o)
1hello (x) → 숫자로 시작 불가능

키워드

- 특정한 의미가 부여된 단어
- 파이썬이 만들어질 때 예약해 놓은 것
- 이미 특정 기능을 수행하고 있기 때문에, 사용자가 이름을 정할 때 키워드를 사용하면 안 됨
- 코드 전용 에디터에서 구분해 줌 (다 외울 필요는 없다)

```
import keyword  
print(keyword.kwlist) # 파이썬에서 사용하는 키워드 출력 명령어
```

4. 연산자

연산자

- 연산자란 데이터를 처리하고 조작하는데 사용되는 기호
- 연산자를 활용하여 값의 계산, 데이터 할당, 비교 등 작업을 수행
- 각 연산자를 역할에 맞게 사용
- 외울 필요없이 코딩을 하다면 자연스럽게 익혀지게 됨

산술연산자

- 수학적 수식을 처리할 때 사용
- **48 / 2 (9 + 3) = 2????? or 288????? or 수식이 잘못**

연산자	연산 작업	설명
+	n1 + n2	더하기
-	n1 - n2	빼기
*	n1 * n2	곱하기
/	n1 / n2	나누기
//	n1 // n2	몫
%	n1 % n2	나머지
**	n1 ** n2	거듭제곱

- 예제

```
print(2 + 5) # 7
print(7 - 4) # 3
print(5 * 6) # 30
print(10 / 3) # 3.33..
print(10 // 3) # 3
print(10 % 3) # 1
print(2 ** 4) # 16
```


연산 순서와 괄호()

- 수학에서 곱셈과 나눗셈이 덧셈과 뺄셈보다 우선순위가 높음
- 프로그래밍에서도 동일하게 연산자 사이에 우선순위가 존재
- () 괄호를 이용해서 명시적으로 우선순위를 정하거나 가독성을 높임

```
print(7 + 9 * 2) # 25  
print(7 + (9 * 2)) # 25  
print((7 + 9) * 2) # 32
```

퀴즈. $100 - 2 / 7 + 9 * 3 = 41.0$ 의 수식이 맞게 ()를 넣어보세요

대입연산자와 복합대입연산자

- 변수에 값을 할당하는 데 사용하는 연산자: =
- 복합대입연산자(val이 변수이름일때)

연산자	연산 작업	설명
+=	val += n	val = val + n
-=	val -= n	val = val - n
*=	val *= n	val = val * n
/=	val /= n	val = val / n
//=	val //= n	val = val // n
%=	val %= n	val = val % n
**=	val **= n	val = val ** n

- 예제

```
num = 5
num += 5
print("+=", num) # 10
num -= 2
print("-=", num) # 8
num *= 4
print("*=", num) # 32
num /= 2
print("/=", num) # 16.0
num //= 3
print("//=", num) # 5.0
num %= 3
print("%=", num) # 2.0
num **= 4
print("**=", num) # 16.0
```

비교연산자

- 참과 거짓을 계산하는 부등식. True와 False로 값이 반환

연산자	연산 작업	설명
>	$n1 > n2$	$n1$ 이 $n2$ 보다 크다
<	$n1 < n2$	$n1$ 이 $n2$ 보다 작다
==	$n1 == n2$	$n1$ 과 $n2$ 가 같다
>=	$n1 >= n2$	$n1$ 이 $n2$ 보다 크거나 같다
<=	$n1 <= n2$	$n1$ 이 $n2$ 보다 작거나 같다
!=	$n1 != n2$	$n1$ 과 $n2$ 가 다르다

- 예제

```
print(2 > 1) # True
print(3 < 2) # False
print(1 == 1) # True
print(3 >= 4) # False
print(5 <= 5) # True
print(3 != 3) # False
```

- 주의) 대입연산자(=)와 비교연산자(==) 사용을 헷갈리지 말것!

논리연산자

- 참과 거짓 사이의 논리적 관계를 평가하는데 사용되는 연산

연산자	연산 작업	설명
and	A and B	두 조건이 모두 참이면 True, 하나라도 거짓이면 False
or	A or B	두 조건이 하나라도 참이면 True, 모두 거짓일때만 False
not	A not B	True/False 값을 반전

```
# 변수 선언
a = 2 > 1
b = 3 < 2
c = 1 == 1
d = 3 >= 4
# and 연산자
print("두 조건 참: ", a and c) # True
print("하나라도 거짓", a and b) # False
# or 연산자
print("하나라도 참:", b or c) # True
print("모두 거짓:", b or d) # False
# not 연산자
print("not a:", not a) # False
print("not b:", not b) # True
```

in 연산자

- 문자열 및 리스트, 튜플, 딕셔너리, 셋(Set)에서 값이 존재하는 여부 확인
- 해당파일 존재시 True반환. 없을시 False반환(not in은 반대)

찾을 값 **in** 찾을 곳

```
a = "Hello World"
print("H" in a) # True
print('h' in a) # false
print('a' not in a) # True
print('l' not in a) # False
```

```
a = ['q', 'w', 'e', 'r', 'w']
print('a' in a) # False
print('q' in a) # True
print('w' not in a) # False
print('j' not in a) # True
```

실습. 연산자 연습

- 숫자하나를 변수에 할당하고 그 숫자가 짝수인지 홀수인지 판별
- 지금껏 배운내용으로만 코드를 작성
- 출력 결과

```
True면 짝수, False면 홀수: False
```

5. 자료형

변수의 자료형

- 자료형은 프로그래밍에서 데이터의 종류와 성질을 나타내는 분류
- 메모리에서 데이터를 어떻게 저장하고 처리할 지를 정의
 - 예) num = 1 과 num = "1" 은 다름
- 변수의 사이즈 알아보는 방법(바이트 단위)

- sizeof(값)

```
from sys import sizeof
print(sizeof(1)) # 28
print(sizeof("1")) # 42
```

- 변수 자료형을 알아보는 방법

- type(값)

```
print(type(1)) # <class 'int'>
print(type(3.14)) # <class 'float'>
print(type('파이썬 기초과정')) # <class 'str'>
print(type(True)) # <class 'bool'>
print(type(None)) # <class 'NoneType'>
```


변수의 자료형

- 변수의 자료형의 종류
- int(정수형) : 소수점이 없는 정수 값
 - 예) `int_type = 10`
- float(실수형) : 소수점을 포함한 실수 값
 - 예) `float_type = 3.14`
- bool(불리언형) : True(참) or False(거짓) 두가지값만 가질 수 있음
 - 예) `is_active = True`
- None(널형) : 값이 없음을 나타냄. 변수이름은 선언하지만 아직 값을 가지지 않을 때 사용(조건식에서 사용할 경우 False)
 - 예) `result = None`

변수의 자료형

- `str`(문자열형) : 문자들의 집합. 홑따옴표 또는 쌍따옴표안에 표기
 - 예) `str_type = '코딩온 파이썬 기초과정'`
- `list`(리스트형) : 여러값을 하나의 변수안에 저장. 순서가 중요
 - 예) `list_type = [1, 2, 3, 4, 5]`
- `tuple`(튜플형) : 리스트와 유사. 생성후 값을 변경할 수 없음
 - 예) `tuple_type = (10.0, 20.0, 30.0)`
- `dict`(딕셔너리형) : 키-값 형태로 데이터를 저장. 키를 통해 값을 조회
 - 예) `dict_type = { 'name' : 'Python', 'type' : 'basic' }`
- `set`(집합형) : 중복을 허용하지 않은 값 모임. 순서가 없음
 - 예) `set_type = { 1, 2, 3, 4, 5 }`

자료형의 형변환

- 이전 실습문제에서 변수에 숫자를 할당하지 않고 input()을 이용하여 숫자를 입력 받게 된다면?

```
num = input("숫자입력 하세요 ")
```

- 위와 같이 코드를 변경 후 실행하면 결과는?

```
a = num % 2
      ^
TypeError: not all arguments converted during string formatting
```

- 오류가 나타나게 됨
- 이는 input()으로 입력한 값은 문자열이기때문
- 정상적으로 사용하려면 input()으로 받은값을 숫자로 변경

자료형의 형변환

- 형변환이란 데이터 타입을 다른 타입으로 변환하는 과정
- 자동 형변환 : 암묵적으로 데이터 타입 변환
 - 정수와 실수 연산 : 정수와 실수가 함께 연산되면 정수가 실수로 자동 변환

```
result = 3 + 2.5 # 5.5
```

- 명시적 형변환 : 개발자가 명시적으로 형변환
 - int() : 실수나 문자열의 숫자를 정수로 변환
 - float() : 정수나 문자열을 숫자를 실수로 변환
 - str() : 문자열로 변환
- 단, 형변환시 데이터 손실이 될 수 있음.
 - 예) 실수를 정수로 형변환시 소수점 이하는 잘려 나감

6. 다양한 문자열 출력

문자열 출력

- 앞으로 다양하게 문자열을 출력해야할 경우가 많이 존재
- 개발 중 변수의 값이나 프로그램의 상태를 출력하여 코드를 올바르게 작성했는지 확인
- 프로그램이 실행될 때 중요한 정보를 출력함으로써 사용자에게 필요한 데이터를 전달하고 안내
- 복잡한 계산이나 로직을 수행한 후, 그 결과를 직관적으로 표현
- 문자열 출력으로 동료 개발자에게 로그나 동작상태 정보를 전달

문자열 연산하기

- + 연산자 : 문자열끼리 연결하기
- * 연산자 : 문자열을 반복해서 출력

```
print("안녕하세요" + " " + "반갑습니다.") # 안녕하세요 반갑습니다.  
# print("문자열과 숫자" + 123) # 오류  
print("hey!" * 10) # hey!hey!hey!hey!hey!hey!hey!hey!hey!hey!  
  
name = "홍길동"  
print("내 이름은 " + name + "입니다.") # 내 이름은 홍길동입니다.
```

여러줄 문자열 출력하기

- 한줄의 문자열이 아닌 여러줄 문자열을 출력하는 방법
- 주석때 사용한 독스트링을 사용
- 독스트링을 변수에 할당하면 문자열로 출력이 가능

```
korea_song = """
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려강산
대한사람 대한으로 길이 보전하세
"""

print(korea_song)
```

```
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려강산
대한사람 대한으로 길이 보전하세
```


따옴표 출력

- 쌍따옴표 또는 홑따옴표의 경우 두개를 조합하여 출력할 수 있음
- 예) " '오늘 저녁 뭐 먹지?' 라고 생각하는 중이다." 를 출력할 때
- print(" '오늘 저녁 뭐 먹지?'라고 생각하는 중이다") 또는 print(" '오늘 저녁 뭐 먹지?'라고 생각하는 중이다") 로 출력 가능

• 예)

```
print("'오늘저녁 뭐먹지?'라고 생각하는 중이다')  
print("'오늘 저녁 뭐먹지?'라고 생각하는 중이다")  
"오늘저녁 뭐먹지?"라고 생각하는 중이다  
'오늘 저녁 뭐먹지?'라고 생각하는 중이다
```

문자열 이스케이프

- 이스케이프 문자 : 프로그램에서 특수한 기능을 수행하기 위한 문자
- 역슬래시(\)로 시작

\n	줄바꿈 (New Line)	print("Hello\nWorld")
\t	탭 (Tab)	print("Hello\tWorld")
\\	역슬래시 (Backslash)	print("This is a backslash: \\")
\'	홀따옴표 (Single quote)	print('It\'s a book')
\"	쌍따옴표 (Double quote)	print("He said \"Hello\"")

```
줄바꿈: Hello
World
탭: Hello      World
역슬래시: This is a backslash: \
홀따옴표: It's a book
쌍따옴표: He said "Hello"
```

문자열 포매팅

- 문자열을 사용하다보면 다양한 방법으로 쓸 경우가 존재
- 예를들어 “올해는 2024년 용띠의 해이다”를 출력
- 여기서 2025년이 되면 2024년을 2025로 변경하고 용띠를 뱀띠로 변경해야한다고 가정
- 그리고 또 2026년이 되면 2026과 뱀을 말로 변경
- 이렇게 계속 특정한 값만 변경해야할 때 쓰는것이 포매팅

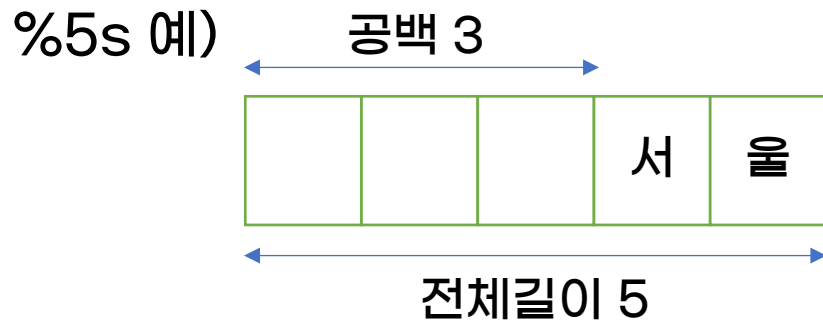
• 예)

```
year = "올해는 %d년 %s의 해이다" % (2024, "용띠")  
print(year) # 올해는 2024년 용띠의 해이다
```

포맷코드를 사용한 문자열 포매팅

- 문자열 % 출력값 형태

```
number = "저는 올해 %d살입니다." % 20
print(number) # 저는 올해 20살입니다.
calc = "10 나누기 4는 %.3f입니다." % 2.5
print(calc) # 10 나누기 4는 2.500입니다.
text = "저는%5s에서 살고있습니다." % "서울"
print(text) # 저는   서울에서 살고있습니다.
char = "이모티콘은 %c 이것으로 할게요" % "😊"
print(char) # 이모티콘은 😊 이것으로 할게요
```



- 사용법

- %d 는 정수
- %f 는 실수
- %s 는 문자열
- %c 는 문자
- %뒤 숫자는 문자열 길이
- %.숫자는 소수점 자리수

format() 사용한 포매팅

- 문자열.format() 형태
- {}안에 숫자나 이름을 넣고 그와 대응되는 값을 넣는 형태
- format()안에 값이 0부터 1:1 대응
- 이름형태로 넣을때는 format()에는 이름=값형태로 작성
- {}안에 숫자를 안넣을때는 format()안에 순서를 지켜서 넣으면 됨

format() 사용한 포매팅

- format()의 다양한 사용법(1)

```
country = "대한민국"
city = "서울"
people = "한국인"
text = "저는 올해 {0}살입니다.".format(20)
print(text) # 저는 올해 20살입니다
text = "저는 {0}사람이며 {1}에 살고있습니다.".format(country, city)
print(text) # 저는 대한민국사람이며 서울에 살고있습니다.
text = "제가 사는 {0}은 {country}에 있습니다.".format(city, country="한국")
print(text) # 제가 사는 서울은 한국에 있습니다
text = "나는 {1}이다. {{ 그리고 }} {0}에 산다.".format(city, people)
print(text) # 나는 한국인이다. { 그리고 } 서울에 산다.
text = "{}점수: {}점, {}점수: {}점".format("영어", 100, "수학", 90)
print(text)
```

format() 사용한 포매팅

- format()의 다양한 사용법(2)

```
a = "[{0:<10}]".format("hey")
print(a) # 좌측정렬, 나머지공백 [hey      ]
a = "[{0:>10}]".format("hey")
print(a) # 우측정렬, 나머지공백 [      hey]
a = "[{0:^10}]".format("hey")
print(a) # 가운데정렬, 좌/우공백 [  hey   ]
a = "[{0:!<10}]".format("hey")
print(a) # 좌측정렬, 특수문자 [hey!!!!!!!]
a = "[{0:^20.7f}]".format(1 / 3)
print(a) # 가운데정렬, 좌/우공백, 전체길이 20 [      0.3333333      ]
a = "[{0:;,}]".format(123456789)
print(a) # 세자리수 마다, [123,456,789]
```

f 문자열 포매팅

- 파이썬 3.6 버전 부터 사용 가능한 기능
- 문자열 앞에 f 접두사를 붙이면 f 문자열 포매팅 기능을 사용 할 수 있다
- 변수 값을 생성한 후에 그 값을 참조할 수 있음
- 표현식 지원 (문자열 안에서 변수와 +,- 같은 수식 함께 사용 가능)
- (<),(>),(^),(,) 등 특수기호 사용법은 format() 사용법과 동일

```
name = '홍길동'
age = 20
text = f'내이름은 {name}입니다. 나이는 {age + 1}살입니다.'
print(text) # 내이름은 홍길동입니다. 나이는 21살입니다.
text = f'내이름은 [{name: !^20}]'
print(text) # 내이름은 [!!!!!!!!!!홍길동!!!!!!!!!!!!]
```


실습. 이스케이프 연습

- 아래 예제와 같이 멍멍이를 출력하세요
- 결과 :

```
| \_ / |  
| q  p |   /}  
(  @  ) "" "" \  
| " ^ " ^      |  
| | _ / = \ \ _ _ |
```

실습. f 문자열 포매팅 실습

- 1) 본인의 이름을 가운데로 정렬하고 = 문자로 공백 채우기
- 2) 아래와 같이 출력하기

```
문자열 실습입니다. { 중괄호 }를 출력해 보세요
```

7. 문자열 관련 함수

문자열 관련 함수

- 앞서 배운 문자열 출력과 함께 다양한 함수들을 이용하여 문자열을 조작 할 수 있음
- 이 함수들로 다양한 상황에서 문자열을 더 정확하게 처리하거나 재생산하여 원하는 방향으로 개발을 진행할 수 있음
- 메서드 = 함수와 동일 한 기능을 하는것들
- 함수는 단독으로 사용이 가능하지만 메서드는 . 으로 연결되어 사용되는 것을 지칭함(메서드에 대한 내용은 추후 더 자세하게 설명)
- 변수명.메서드명() 형태

문자열 인덱싱

- 문자열 인덱싱: 문자열의 각 문자를 특정 위치를 통해 접근하는 방법
 - 인덱싱 : 특정 위치의 요소를 접근하거나 참조하기 위해 인덱스 번호를 사용하는 작업

H	e	l	l	o	,		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11	12

← 인덱스

- 모든 인덱스는 앞에서는 항상 0부터 시작. 뒤에서는 -1부터 시작
- `a = "Hello, Python"` 에서 P값에 접근하는 방법은 식별자[인덱스]
- P값에 접근 : `a[7]` or `a[-6]`

퀴즈. 위 Python을 인덱싱을 이용해서 출력해보세요.

문자열 슬라이싱

- 문자열 슬라이싱: 문자열의 특정 부분을 추출하는 방법(잘라냄)
- 슬라이싱 구문
 - 식별자[start:end]
 - start: 문자열의 시작. 잘라내는 부분이 포함됨
 - end: 문자열의 끝. 잘라내는 부분이 포함되지 않음
 - start(0부터 시작할때)나 end(끝까지)값은 생략 할 수 있음.
- 이전 퀴즈에서 슬라이싱을 적용한다면? d[7:]

퀴즈. 20240930을 년,월,일로 출력해보세요

길이 구하기

- 문자열 길이 구하기(빈문자도 문자열에 포함)
 - len() 이용

```
a = "Hello, Python"  
print(len(a)) # 13
```

개수 세기

- 문자 개수 세기
 - count(찾을문자)

```
a = "Hello, Python"  
print(a.count('l')) # 2
```


위치 찾기

- 문자가 어디에 있는지 위치 찾기1(다수일경우 가장 앞선 문자위치. 없으면 -1)

- find(찾을문자)

```
a = "Hello, Python"
print(a.find('o')) # 4
print(a.find('m')) # -1
```

- 문자가 어디에 있는지 위치 찾기2(다수일경우 가장 앞선 문자위치. 없으면오류)

- index(찾을문자)

```
a = "Hello, Python"
print(a.index('o')) # 4
print(a.index('m')) # ValueError: substring not found
```

바꾸기, 나누기

- 문자열 바꾸기

- replace(변경하고자 하는문자, 변경할 문자)

```
a = "Hello, Python"
print(a.replace("Hello", "안녕")) # 안녕, Python
```

- 문자열 나누기(나누고싶은문자는 사라짐), 리스트로 결과 표시

- split(나누고싶은문자)

```
a = "Hello, Python"
print(a.split(',')) # ['He', '', 'o, Python']
```

대/소문자, 공백지우기

- 대,소문자로 바꾸기
 - 대문자: upper()
 - 소문자: lower()

```
a = "Hello, Python"
print(a.upper()) # HELLO, PYTHON
print(a.lower()) # hello, python
```

- 공백지우기
 - 오른쪽: rstrip()
 - 왼쪽: lstrip()
 - 양쪽: strip()

```
a = "    Hello    "
print "["+a.rstrip()+"]" # [    Hello]
print "["+a.lstrip()+"]" # [Hello    ]
print "["+a.strip()+"]"  # [Hello]
```

문자열 판별하기(숫자판별)

- `isdecimal()` : 0-9 범위의 10진수 숫자만 허용. 가장 제한적
- `isdigit()` : 10진수 및 일부 유니코드 숫자 포함. `isdecimal()`보다 넓은 범위
- `isnumeric()` : 모든 유니코드 숫자와 분수 등을 포함. 가장 포괄적

```
print("123".isdecimal())      # True
print("123.45".isdecimal())   # False (소수점 포함 시 False)
print("23".isdecimal())       # False (지수 표현도 False)
print("123".isdigit())        # True
print("23".isdigit())         # True (제곱 기호 등 유니코드 숫자)
print("123.45".isdigit())     # False (소수점 포함)
print("123".isnumeric())      # True
print("23".isnumeric())       # True (제곱 기호 등 유니코드 숫자)
print("123.45".isnumeric())   # False (소수점 포함)
print("一二三".isnumeric())    # True (중국어 숫자)
print("½".isnumeric())        # True (분수 기호)
```

문자열 판별하기(문자,공백판별)

- isalpha(): 문자열이 알파벳 문자로만 구성되어 있는지 확인
- isalnum(): 문자열이 알파벳 또는 숫자로만 구성되어 있는지 확인
- isspace(): 문자열이 공백 문자로만 구성되어 있는지 확인

```
print("hello".isalpha())      # True
print("안녕하세요".isalpha()) # True
print("hello123".isalpha())   # False (숫자가 포함되어 있으므로 False)
print("hello123".isalnum())   # True
print("hello!".isalnum())     # False (특수 문자 포함)
print("안녕하세요123".isalnum()) # True
print("   ".isspace())        # True (공백 문자만 포함)
print("\t\n".isspace())       # True (탭과 줄바꿈 포함)
print("hello".isspace())      # False
```

문자열 판별하기(대소문자 판별)

- islower() 문자열이 모두 소문자로 구성되어 있는지 확인
- isupper() 문자열이 모두 대문자로 구성되어 있는지 확인
- istitle() 문자열이 각 단어의 첫 글자가 대문자인지 확인(띄어쓰기 기준)

```
print("hello".islower())      # True
print("Hello".islower())     # False (대문자 포함)
print("hello123".islower())  # True (숫자는 검사에서 제외)
print("HELLO".isupper())     # True
print("Hello".isupper())     # False (소문자 포함)
print("HELLO123".isupper())  # True (숫자는 검사에서 제외)
print("Hello World".istitle()) # True
print("Hello world".istitle()) # False (두 번째 단어가 소문자)
print("안녕하세요".istitle()) # False (한글은 대소문자 구분이 없어 False)
```

실습. 종합실습

- 입출력 실습 - 다음과 같이 실행되도록 코드를 작성하세요.

1번) 이름을 입력하세요. 홍길동
나이를 입력하세요. 100
안녕하세요! 홍길동님 (100세)

2번) 이름을 입력하세요. 홍길동
태어난 년도를 입력하세요. 2010
올해 년도를 입력하세요. 2023
올해는 2023년, 홍길동님의 나이는 14세 입니다

학습정리

- 변수는 데이터를 담는 그릇이며 변수를 활용해서 프로그래밍을 진행 할 수 있다.
- 변수에 값을 할당할 수 있으며 언제든지 재할당이 가능하다.
- 상수는 재할당하지 않는 값을 넣을때 사용한다.
- 산술연산자를 활용하여 수학적 계산을 할 수 있으며 ()를 이용하여 연산의 우선순위를 정할 수 있다.

학습정리

- 비교연산자와 논리연산자, in연산자를 활용하여 True, False를 출력할 수 있다. 이는 추후 배울 조건식에서 매우 중요하게 사용된다.
- 문자열끼리도 연산이 가능하며 최근 파이썬 3.6이상부터는 f문자열 포매팅을 활용하여 문자열을 출력한다.
- 인덱스는 0부터 시작한다.
- 이름()은 함수이며 변수.이름()은 메서드라고 불리운다.

학습정리

- 함수 `len()`은 길이, 메서드인 `count()`는 문자개수세기, `find()`와 `index()`는 문자의 첫번째 인덱싱값 찾기, `replace()`는 문자열바꾸기, `split()`은 문자열 나누기에 사용된다.
- `find()`는 문자 존재여부 확인시 유용하며 `index()`는 반드시 문자가 존재한다고 확신할 때 사용한다.

다음 수업은?

- 이번수업은 앞으로 파이썬에 사용할 다양한 개념에 대해서 알아보았습니다.
- 다음시간에는 파이썬의 자료형에 대해서 공부하도록 하겠습니다.

복.습.철.저

수고하셨습니다