

1강

파이썬 프로그래밍 기초

컴퓨터의 이해



>>> 컴퓨터과학과 정재화 교수

>>> 01

데이터와 정보

Python Programming

데이터의 개념

👉 어떤 현상이나 사실에 대한 설명 또는 설명의 집합



데이터의 정의

데이터 (data) 명사

(국립국어원 표준국어 대사전 참조)

- ① 관찰이나 실험, 조사로 얻은 사실
- ② 이론을 세우는데 기초가 되는 사실 또는 바탕이 되는 자료

불다

밝다

뜨겁다

3500도

질적 데이터

양적 데이터

관찰이나 측정을 통해 얻은 수치, 문자 형태의
표현할 수 있는 질적(quality) 또는 양적(quantity) 값



정보의 개념

👉 문제 또는 질문을 해결하기 위해 사용할 수 있는 데이터와 데이터의 집합 → 의미가 있는 데이터



저 파란색은 무엇일까?

붉고 뜨거운 것과
무엇이 다른것일까?



정보의 정의

정보 (information) 명사 (국립국어원 표준국어 대사전 참조)

- ① 어떤 데이터나 소식을 통하여 얻은 지식이나 상태의 총량
- ② 관찰이나 측정을 통하여 수집한 자료를 문제 해결에 도움이 될 수 있도록 정리한 지식

관찰과 측정을 통해 얻은 데이터를
처리(정렬, 합산, 군집화 등)하여 실제 문제 해결에
도움이 되는 데이터 또는 결과물

정보 처리 과정

- ☞ 데이터는 **현상**에 대한 **관찰**과 **측정**으로 생성
- ☞ 데이터는 기록된 사실이지만, 직접적으로 문제를 해결하는데 도움이 되지 **않음**
- ☞ **처리**와 **가공**을 통해 정보로 변환



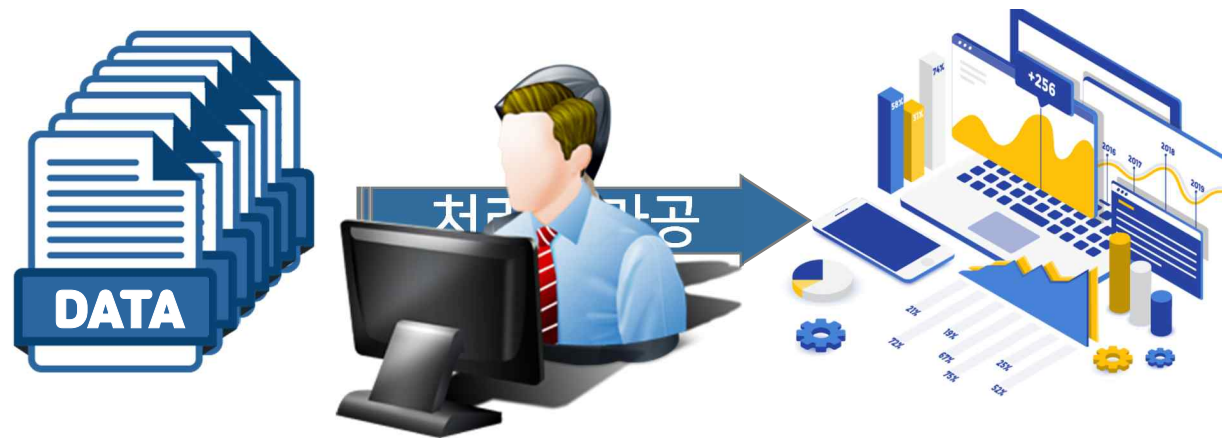


컴퓨터의 개념

Python Programming

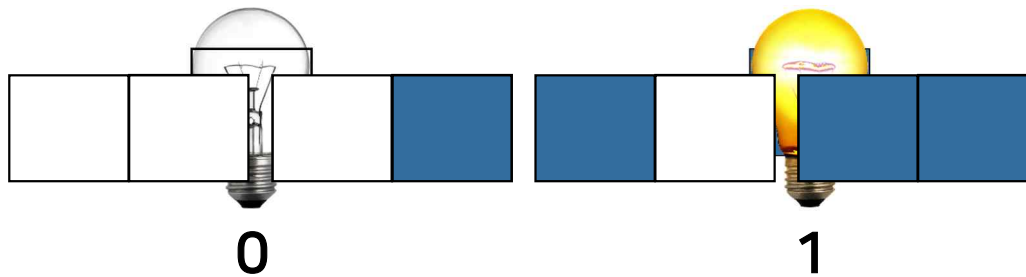
컴퓨터의 정의

☞ 정의된 방법에 따라 입력된 데이터를
자동으로 처리하여 정보를 생산하는 기계

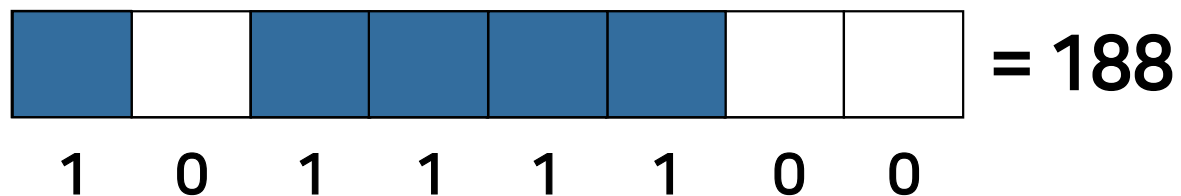


데이터의 기본 단위

➡ 비트(bit, binary digit)



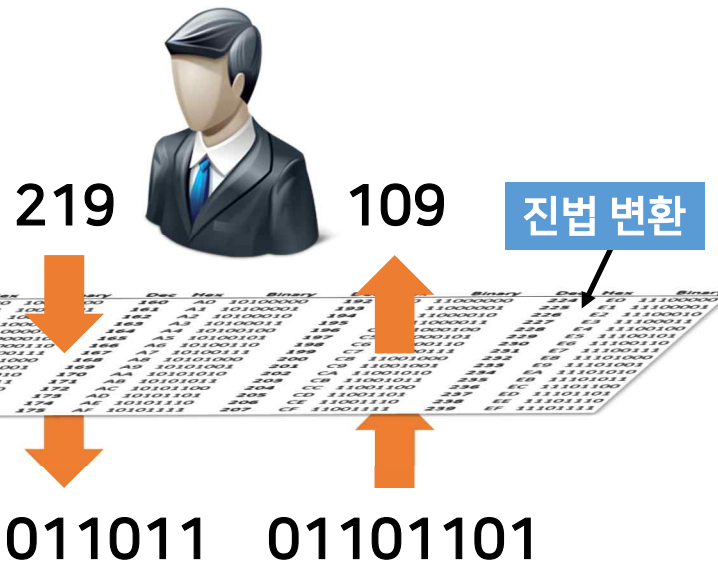
➡ 바이트(byte)



02. 컴퓨터의 개념

숫자 데이터의 표현

10진수, 12진수 등

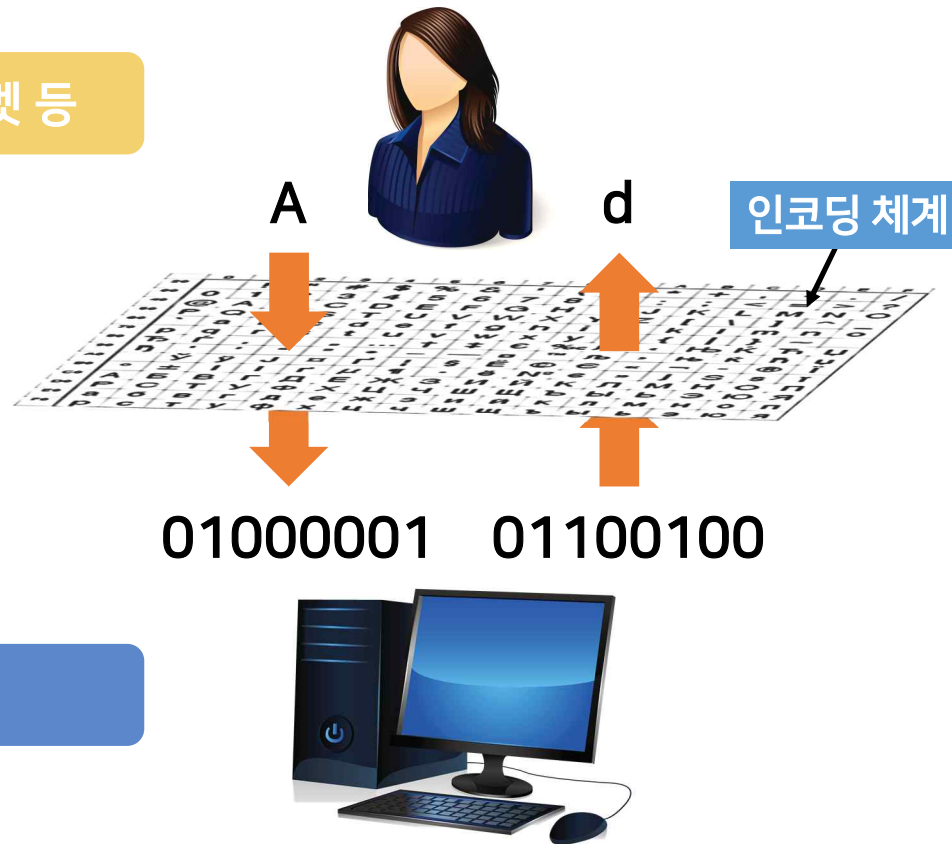


2진수



문자 데이터의 표현

한글, 알파벳 등



2진수

02. 컴퓨터의 개념



ASCII 코드표

2진법	십진법	문자	2진법	십진법	문자	2진법	십진법	문자
010 0000	32	SP	100 0000	64	@	110 0000	96	`
010 0001	33	!	100 0001	65	A	110 0001	97	a
010 0010	34	"	100 0010	66	B	110 0010	98	b
010 0011	35	#	100 0011	67	C	110 0011	99	c
010 0100	36	\$	100 0100	68	D	110 0100	100	d
010 0101	37	%	100 0101	69	E	110 0101	101	e
010 0110	38	&	100 0110	70	F	110 0110	102	f
010 0111	39	'	100 0111	71	G	110 0111	103	g
010 1000	40	(100 1000	72	H	110 1000	104	h
010 1001	41)	100 1001	73	I	110 1001	105	i
010 1010	42	*	100 1010	74	J	110 1010	106	j
010 1011	43	+	100 1011	75	K	110 1011	107	k
010 1100	44	,	100 1100	76	L	110 1100	108	l
010 1101	45	-	100 1101	77	M	110 1101	109	m
010 1110	46	.	100 1110	78	N	110 1110	110	n



컴퓨터와 프로그램

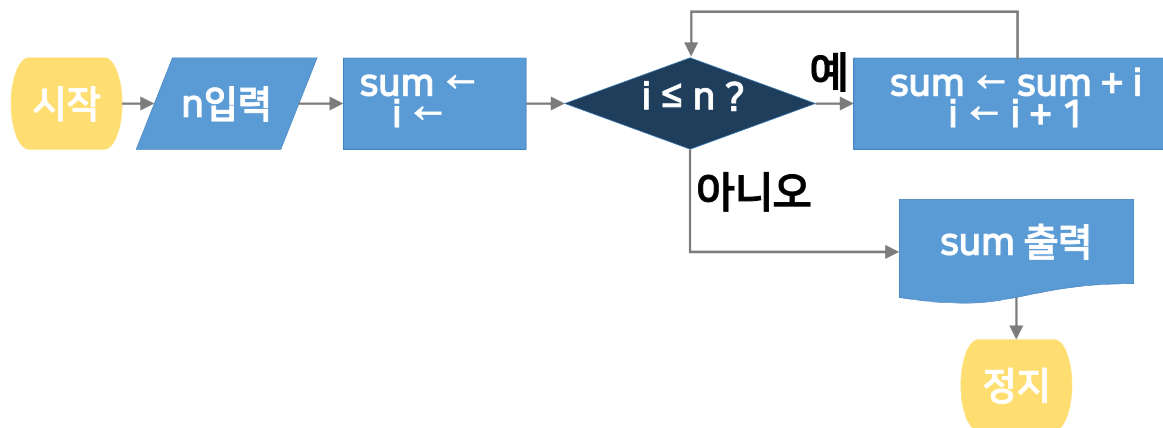
Python Programming

프로그램의 이해

- ➡ 컴퓨터는 정보를 처리하기 위한 방법과 과정을
자의적으로 결정 불가능
- ➡ 처리 방법과 과정이 정의된 프로그램을 사용
 - 프로그램이란 컴퓨터가 어떠한 작업을 자동으로 처리
할 수 있도록 처리 방법 및 순서를 컴퓨터가 이해할 수
있는 언어 형태로 기술한 것
 - 유사한 유형의 여러 문제를 추상화시킨
알고리즘(algorithm)을 구현한 결과물

알고리즘의 정의

- 👉 문제를 풀기 위한 단계별 절차를 체계적 명령의 형태로 기술한 것
- 👉 주어진 명령어를 처리하는 컴퓨터에게 문제를 해결하도록 만드는 정형화된 절차



프로그래밍 언어

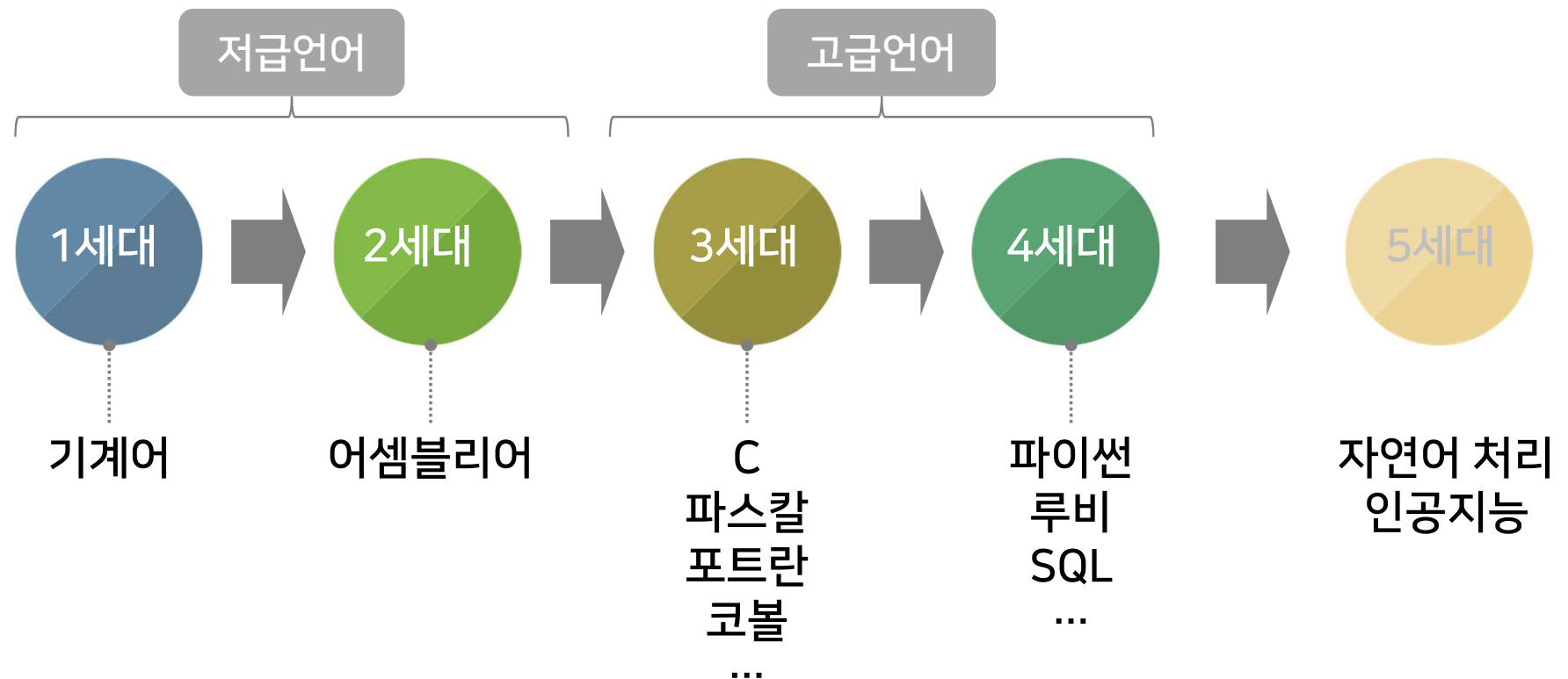
- ➡ 사람과 컴퓨터 사이의 의사소통 도구
- ➡ 프로그래머는 컴퓨터가 이해할 수 있는 언어를 사용하여 프로그램을 작성



작업처리 지시



프로그래밍 언어의 분류



프로그램의 실행

- ☞ 고급 언어로 작성된 프로그램은 기계어 밖에 인식할 수 없는 컴퓨터에 의해 **실행 불가능**
→ **컴파일러** 또는 **인터프리터**를 사용

101010001110001110
101010101010001101
101011011011010011
010101001111100101
010001100011110110

```
while (i <= n) :  
    sum = sum + i  
    i = i + 1  
print("결과는", str
```



컴파일러, 인터프리터



프로그램의 실행

컴파일

- '무엇인가를 모아서 묶음으로 만든다' 의미
- 컴파일러는 프로그램 전체를 입력 받아 모두 기계어로 미리 번역
- 기계가 인식, 실행할 수 있는 형태로 변환(Linking)

인터프리터

- 프로그램 실행 시 한 번에 한 문장씩 실시간으로 읽고 실행

리뷰

파이썬 프로그래밍 기초

하드웨어와 소프트웨어

>>> 컴퓨터과학과 정재화 교수

컴퓨터의 구성요소

하드웨어



컴퓨터를 구성하고 있는
물리적 부품

소프트웨어

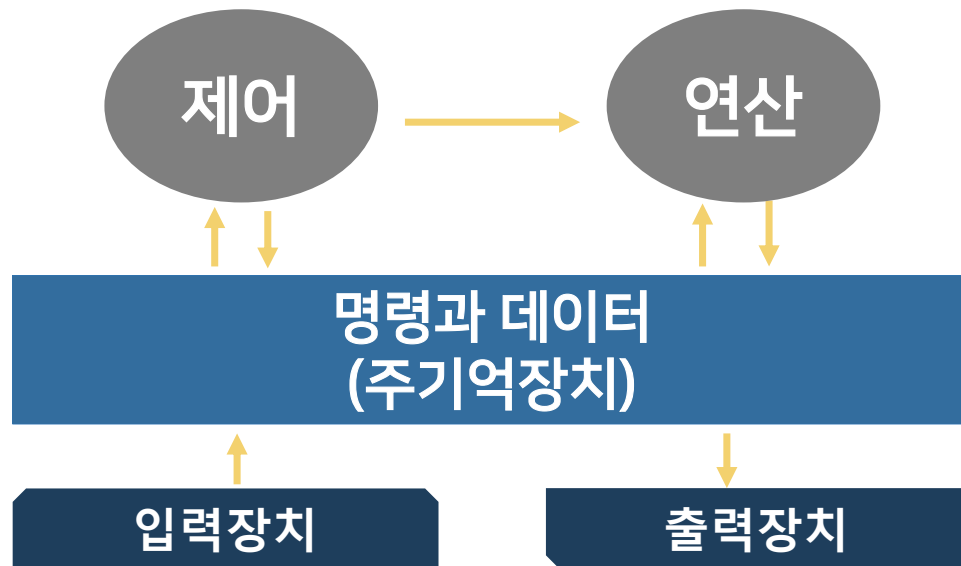


다양한 장치들을
동작시켜 특정 작업을
해결하는 프로그램



하드웨어

👉 기능에 따라 입력장치, 기억·저장장치, 제어장치, 연산장치, 출력장치로 구분



폰 노이만 구조



소프트웨어



소프트웨어는 각각의 고유한 기능을 수행하는 하드웨어가 언제 어떻게 동작하여 문제를 해결할 수 있는지 지시하는 명령어 집합

→ 응용 소프트웨어: 사용자의 업무나 목적에 맞게 문제 해결을 위한 처리 절차를 표현한 명령어 집합

→ 시스템 소프트웨어: 하드웨어를 제어·관리하여 응용 소프트웨어를 실행할 수 있는 환경을 제공

입력장치

- ➡ 명령과 데이터를 컴퓨터에 전달하는 장치
- ➡ 컴퓨터가 처리할 수 있는 2진수 형태로 변환



명령과 데이터



02. 하드웨어의 이해

입력장치의 종류



키보드



마우스

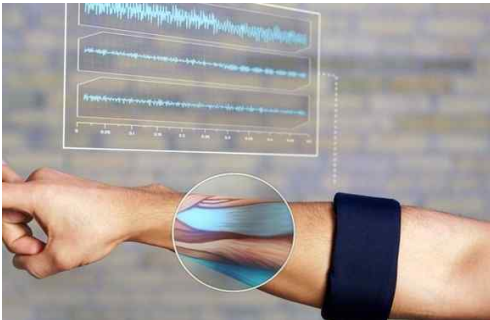


스캐너



터치스크린

특수 입력장치



마이오
(Myo)



립 모션
(Leap Motion)

출력장치

☞ 정보 처리 결과를 인간이 인식 가능한 형태의 데이터로 내보내는 장치

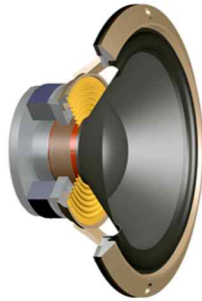
→ 2진수 형태의 데이터를 문자, 숫자, 도형, 음성, 영상 등의 형태로 변환



출력장치의 종류



모니터



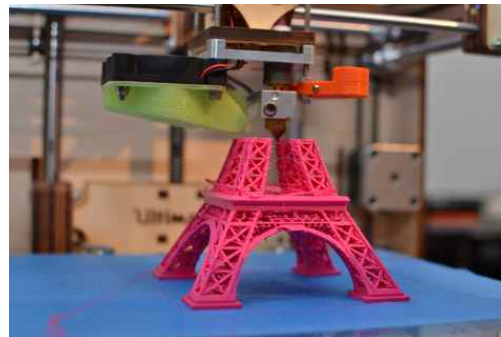
스피커



프린터



초 지향성 스피커



3D 프린터

02. 하드웨어의 이해

특수 출력 장치



HUD
(Head-Up Display)



HMD
(Head-Mounted Display)



기억(저장) 장치



명령과 데이터를 기억(저장)하는 하드웨어



역할에 따라 주기억장치 보조기억장치로 구분

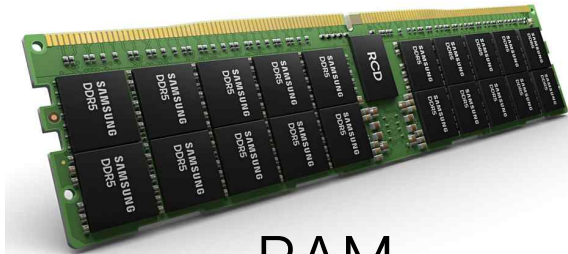
항목	기억장치 (주기억장치)	저장장치 (보조기억장치)
접근속도	빠름	느림
제조단가	높음	낮음
용량	작음	큼
전원 차단 시	ROM: 기억내용 보존 RAM: 모든 내용 초기화	기억내용 보존

02. 하드웨어의 이해

기억(저장)장치의 종류



ROM



RAM



플로피디스크



자기디스크



광학디스크



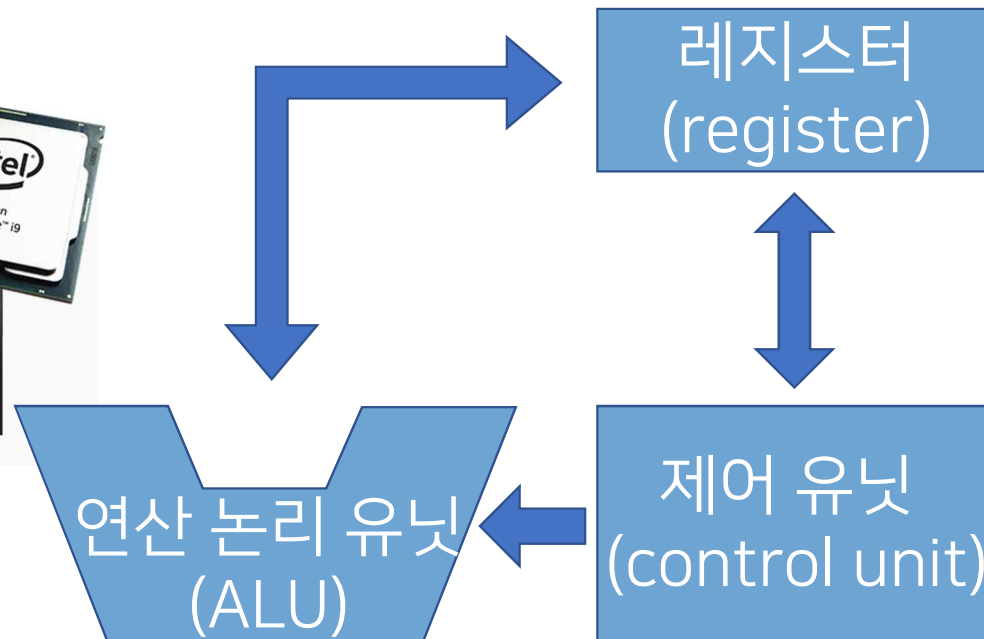
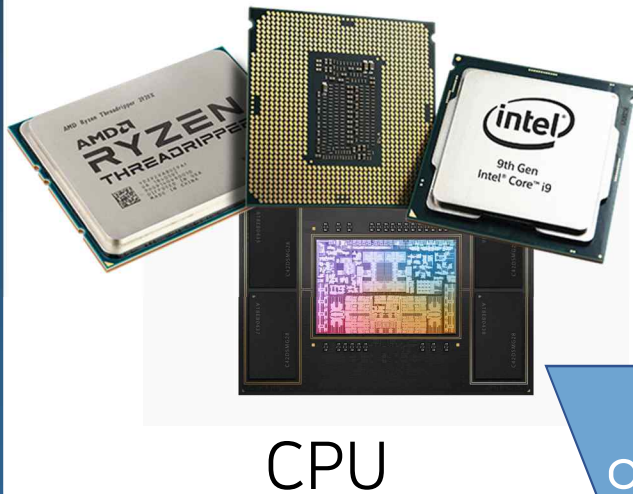
플래시 드라이브



SSD

제어와 연산 장치

👉 명령어와 데이터를 읽고 데이터를 연산 또는 처리하는 장치



3강

파이썬 프로그래밍 기초

파이썬의 이해



>>> 컴퓨터과학과 정재화 교수

>>> 01

파이썬의 개요

Python Programming



파이썬의 탄생 1



- ➡ **히도 판로섬(Guido van Rossum) 1991년 개발**
 - 네덜란드 암스테르담 대학에서 컴퓨터 전공
 - 좋아하는 코미디 'Monty Python's Flying Circus'를 따라 명명
 - 크리스마스 주 연구실이 잠겨 할 일이 없어 만든 프로그래밍 언어
- ➡ **분산 운영 체제(아메바)의 시스템 관리를 위한 셸 스크립팅 언어로 개발**

파이썬의 탄생 2

다중 프로그래밍 패러다임 채용

→정의: 프로그램을 생성하는 접근 방식

→명령형 프로그래밍, 절차적 프로그래밍,
객체지향 프로그래밍, 함수형 프로그래밍 지원

다목적 활용

→응용 프로그램과 웹, 백 엔드 개발, 사물 인터넷
분야 뿐만 아니라 교육적인 목적으로도 활용

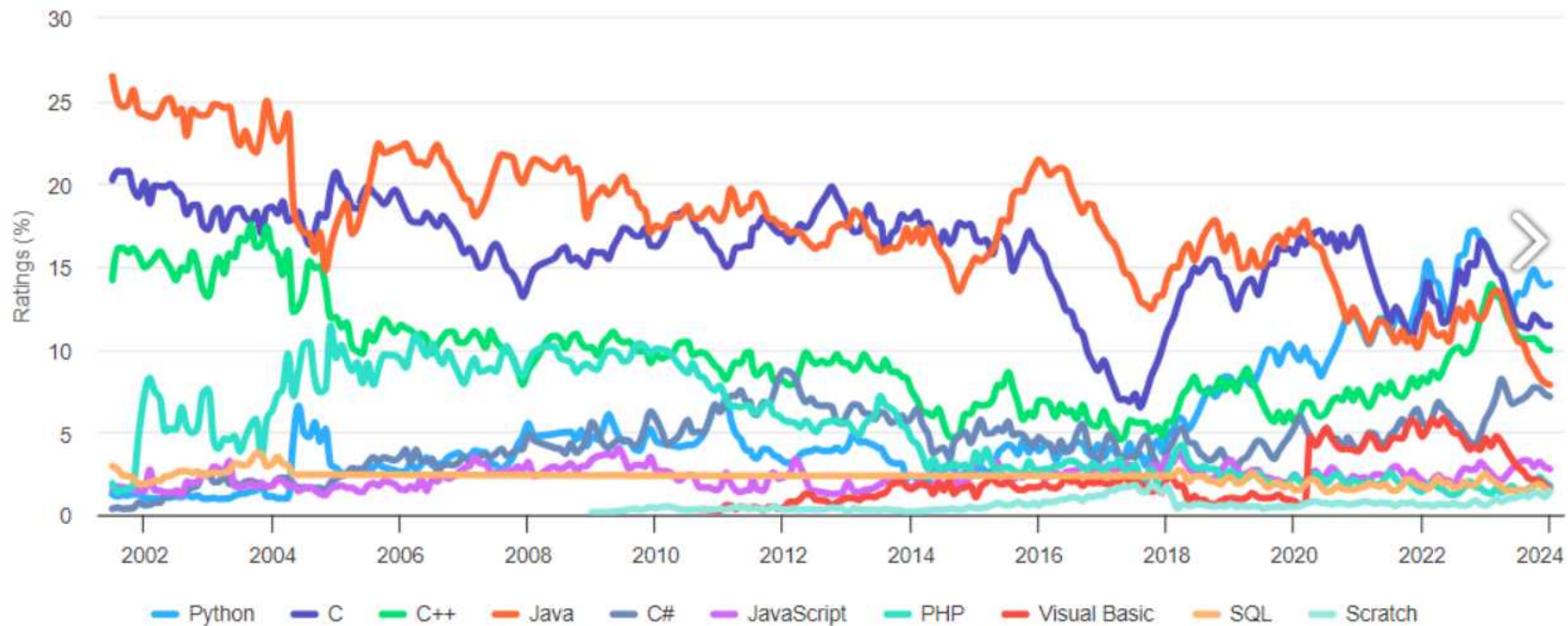
파이썬의 발전 과정

- 👉 1991년 ABC의 후속 프로그래밍 언어로 시작
- 👉 1999년 DARPA에 'Computer Programming for Everybody' 제안
- 👉 2000년 파이썬 2.0 출시
 - 커뮤니티를 통한 개발 체계 시작
- 👉 2008년 파이썬 3.0 출시
 - 비 하위 호환성을 갖는 메이저 업데이트

01. 파이썬의 개요

파이썬의 인기

TIOBE programming community index

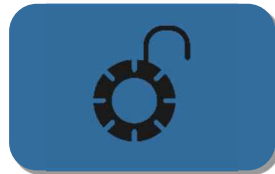


01. 파이썬의 개요

Pythonic



독립적



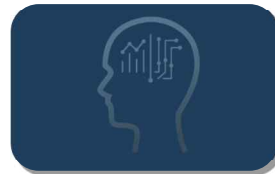
오픈소스



인간적



신속성



직관적

오픈소스

파이썬 관련 개선을 위한 제안(PEP)

- 많은 개발자의 의견을 수용하고 토론하며 발전한 언어
- 새로운 파이썬의 기능, 파이썬 프로세스, 환경에 대해 커뮤니티에 설계 문서나 정보를 제공
- 파이썬 기능의 간결한 기술적 사양과 기능을 위한 근거들을 제공
- 커뮤니티의 의견을 수집, 합의 도출, 반대의견 청취

PEP 8(스타일 가이드) 대표적



인간적 & 직관적



실행할 수 있는 의사 코드(Executable pseudocode) 수준의 문법

```
if 3 in [1,3,5,7]: print("3이 들어있습니다")
```



리스트 [1, 2, 3, 4]에 3이 포함되어 있으면
“ 3이 들어있습니다”를 출력하시오.



생산성 & 신속성

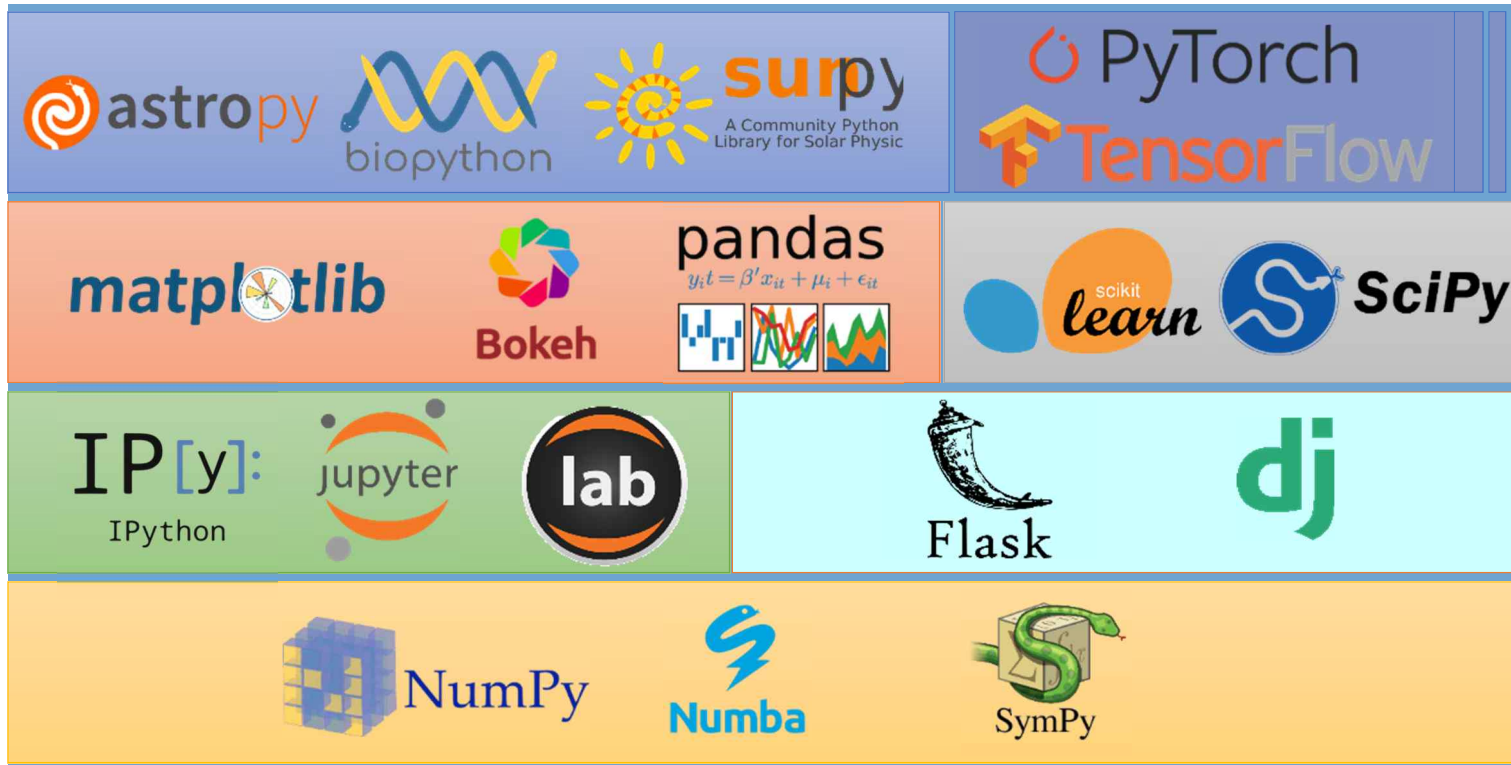
```
int i, n;  
int sum = 0  
  
printf("입력:");  
scanf("%d", &n);  
  
for(i = 0; i < n; i++)  
{  
    sum += i;  
}  
  
print("합은 %d", sum);
```

C

```
n = int(input("입력:"))  
sum = 0  
  
for i in range(1, n+1)  
    sum += i  
  
print("합은" + sum)
```

Python

다양한 커뮤니티와 라이브러리



파이썬의 단점

- 👉 C나 자바 등으로 작성된 프로그램보다 느린 속도
- 👉 완전한 애플리케이션 단독 개발이 불가능
 - 웹 스크립트 언어 용으로 개발
 - 모바일 앱 등 응용 애플리케이션 개발 불가능

 Rust 또는 Go 고려

>>> 02

파이썬 프로그램의 실행

Python Programming

파이썬 실행 환경

- 👉 플랫폼에 독립적이며 인터프리터식 객체지향적, 동적 타이핑(dynamically typed) 대화형 언어
 - 윈도우, 리눅스, 유닉스, 맥OS 등 다양한 운영체제 (플랫폼)에서 별도의 컴파일 없이 실행 가능
 - CPython, PyPy, Cython, Jython 등 다양한 인터프리터 환경 사용 가능
 - 프로그램을 객체로 모델링
 - 변수의 자료형을 지정하지 않음
 - 작성한 코드에 대한 수행 결과를 바로 확인하고 디버깅하면서 코드 작성 가능

CPython

👉 C 언어로 개발된 파이썬 인터프리터

→ C 로 구현한 라이브러리와 연동을 통한 확장에 최적

컴파일러의 유형

① 셀프 호스팅 컴파일러: 부트스트래핑 단계를 통해 자신의 언어로 작성한 컴파일러

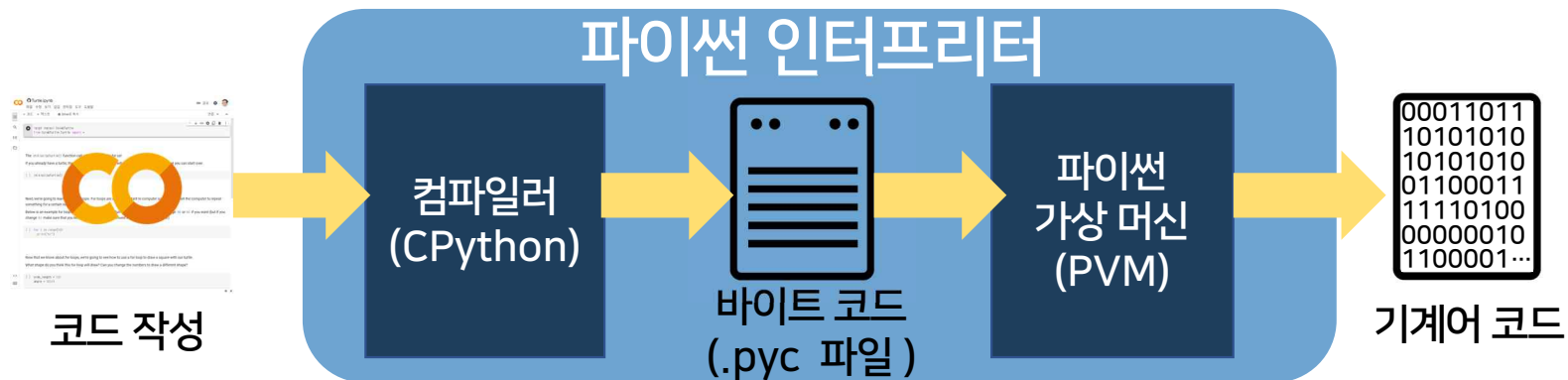
② 소스 대 소스 컴파일러: 타 언어로 작성한 컴파일러

👉 오픈소스로 커뮤니티의 기여로 지속적 발전

→ <https://github.com/python/cpython>

파이썬 프로그램 실행과정

- 👉 파이썬 애플리케이션은 소스 코드 형태로 배포
 - CPython이 컴파일후 바이트코드 .pyc 파일 생성
 - 파이썬 가상머신은 바이트코드를 한 라인 씩 실행
 - 변경없이 재실행 시 바이트코드로 빠르게 실행





파이썬 프로그래밍 환경

Python Programming

03. 파이썬 프로그래밍 환경



Python

python™

About Download

```
# Python 3: List comprehension
>>> fruits = ['Banana', 'Apple', 'Lemon']
>>> loud_fruits = [fruit.upper() for fruit in fruits]
>>> print(loud_fruits)
['BANANA', 'APPLE', 'LEMON']

# List and the enumerate function
>>> list(enumerate(fruits))
[(0, 'Banana'), (1, 'Apple'), (2, 'Lemon')]
```

Get Started

Whether you're new to programming or an experienced developer, it's easy to learn and use Python.

IDLE Shell 3.10.7

File Edit Shell Debug Options Window Help

```
>>> import random
>>> print("Let's play rock-paper-scissors!")
...
Let's play rock-paper-scissors!
>>>
... # Ask the user to choose between rock, paper, or scissors
... user_choice = input("Enter your choice (rock/paper/scissors): ").lower()
...
Enter your choice (rock/paper/scissors): 1
>>>
... # Define the computer's choices
... computer_choices = ['rock', 'paper', 'scissors']
... computer_choice = random.choice(computer_choices)
...
SyntaxError: multiple statements found while compiling a single statement
>>> computer_choices = ['rock', 'paper', 'scissors']
>>> computer_choice = random.choice(computer_choices)
>>> if user_choice == computer_choice:
...     result = "Tie!"
... elif (user_choice == "rock" and computer_choice == "scissors") or \
...       (user_choice == "paper" and computer_choice == "rock") or \
...       (user_choice == "scissors" and computer_choice == "paper"):
...     result = "You win!"
... else:
...     result = "Computer wins!"
>>>
```

Ln: 31 Col: 4

Python source code and installers are available for download for all versions!
Latest: Python 3.11.2

Documentation for Python's standard library, along with tutorials and guides, are available online.
docs.python.org

Looking for work or have a Python related position that you're trying to hire for? Our **relaunched** **community-run job board** is the

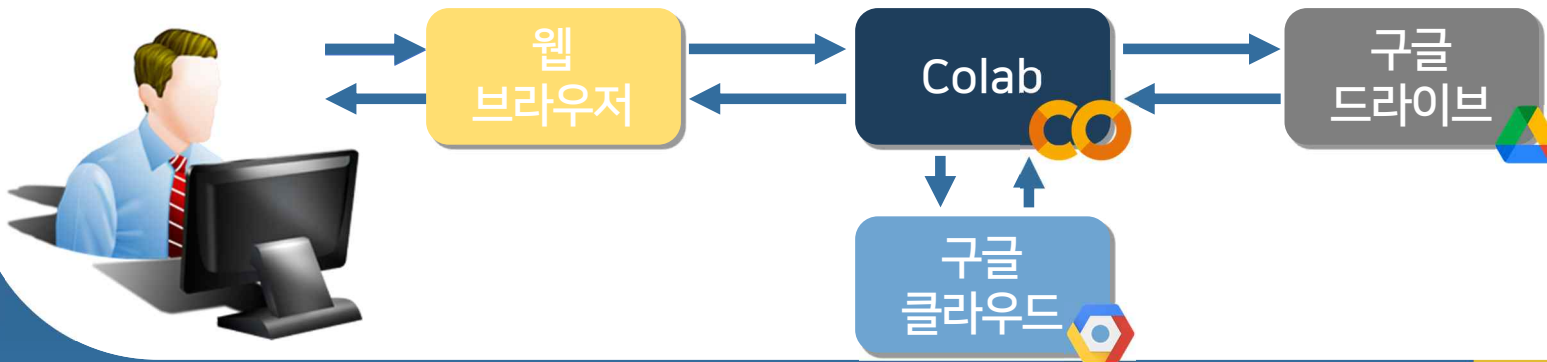
주피터 노트북

오픈소스 기반의 웹 플랫폼

- 파이썬을 비롯한 40여개의 프로그래밍 언어 지원
- 전통적인 소스코드-컴파일-실행 방식에서 벗어나 웹 기반 대화형 개발 및 실행 환경
- 문서화하여 다른 사람과 공유하기가 편리
- 마크다운(Markdown)을 이용하여 코드 관련 타이틀, 설명 등 작성 가능

구글 Colab

- 👉 2017년 과학 연구와 교육을 목적으로 개발
- 👉 클라우드 기반 주피터 노트북 개발 환경
 - 주피터 노트북 + 구글 드라이브를 결합한 서비스
 - 데이터 분석 및 딥러닝 연산 등 고성능 컴퓨팅 리소스 활용 가능



4강

파이썬 프로그래밍 기초

파이썬 시작하기



>>> 컴퓨터과학과 정재화 교수

>>> 01

프로그래밍 기초

Python Programming

숫자와 문자

숫자

- 정수(integer): 소수점이 없는 숫자
- 실수(floating point): 소수점이 포함된 숫자

문자

- 유니코드(unicode) 기반 문자 또는 문자열
- 인용 부호 " 또는 '를 사용하여 표현



기본연산자와 표현식



피연산자와 연산자를 이용한 표현식은 파이썬 인터프리터에 의해 자동 계산

연산자	기능
+	더하기
-	빼기
*	곱하기
/	나누기
**	지수(거듭제곱)

`2 * (7 + 15)`

`2 ** 10 * 5`

`"computer" + "science"`

함수(function)

- ➡ 특정 작업을 수행하는 코드의 집합으로
함수의 이름만으로 실행할 수 있는 단위
→ print 함수: 화면에 데이터를 출력하는 작업

➡ 함수의 기본 구조

```
print("Hello World!")
```

함수이름

입력값(파라미터)



함수의 실행

"Hello World!"

47

print(x) 화면에 *x* 출력

화면에 "Hello World!"를 출력

50

들여쓰기

- 👉 파이썬은 들여쓰기에 의존적 언어
 - 프로그래밍 언어에서는 가독성 향상 목적
 - 코드의 논리적 집합인 블록을 표현
- 👉 들여쓰기는 스페이스 4칸을 권장(PEP 8)
- 👉 블록 중첩 시 추가적인 4칸 들여쓰기 삽입

```
print("Hello World!")  
print("Python is fun")
```

```
print("Hello World!")  
    print("Python is fun")
```

문서화

주석(comment) 사용

→가독성 증대로 개발 속도 향상, 유지보수 용이

주석의 종류

→한 라인 주석에 #을 사용

→여러 라인 주석에 """ 또는 " 3개를 연달아 사용

```
"""성적 계산
#성적 계산
#성적 계산 방식 sum(성적 * 학점)/총학점"""
print((10.5 * 2 + 2 * 3) / (130 - 66))
```



데이터 저장

Python Programming

원뿔의 부피 계산 프로그램

원뿔의 부피 계산 알고리즘

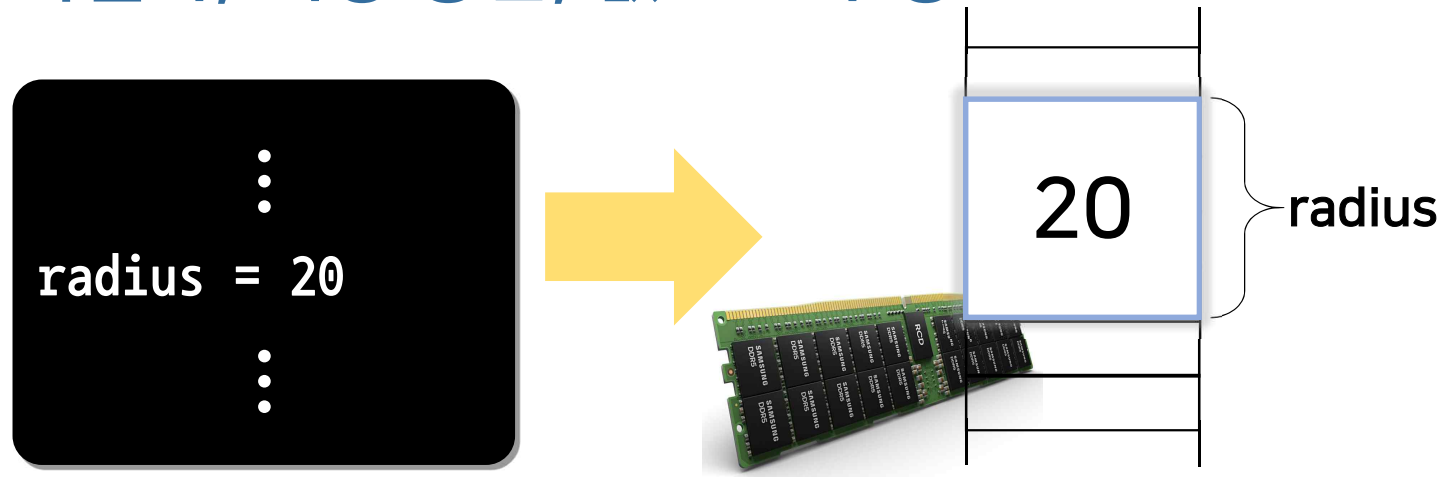
$$\text{부피} = 1/3\pi r^2 h$$

$$\text{겉넓이} = \pi r^2 + \pi r h$$

```
#반지름 20, 높이 30
#부피 출력
print(1 / 3 * 3.14 * 20 ** 2 * 30 )
#겉넓이 출력
print(3.14 * 20 ** 2 + 3.14 * 20 * 30)
```

변수(variable)

- ➡ 프로그램이 실행되는 과정에서 처리되는 값이 어떤 행위(입력, 연산 등)에 따라 그 값이 변할 수 있는 메모리 내의 저장 공간
- ➡ 식별자, 저장 공간, 값으로 구성



식별자

👉 프로그램 내부에 정의된 객체(변수, 함수 등)의 이름

- 문자, 숫자, 밑줄로 구성
- 문자 또는 밑줄로만 시작 가능
- 예약어와 동일할 수 없음
- 길이 제한이 없음

```
volume, Surface, _50, x,  
y, __name
```

```
$D$, 1-a, y, else,  
class, r #R
```



예약어(reserved word)



👉 파이썬 인터프리터에 의해 이미 문법적인 용도로 사용되어 식별자로 사용이 불가능한 단어

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

값의 할당

명령형 패러다임 언어의 특징

- 처리할 데이터와 처리된 결과를 임시적 저장
- 변수의 값을 변경하는 할당문

<u>radius</u>	=	<u>20</u>
		
lvalue		rvalue
(지속되는 대상)		(임시적인 대상)

변수의 사용

```
#반지름, 높이 값 할당
```

```
radius = 20
```

```
height = 30
```

```
#부피 출력
```

```
print(1 / 3 * 3.14 * radius * radius * height)
```

```
#겉넓이 출력
```

```
print(3.14 * radius * radius + 3.14 * radius * height)
```

lvalue

값 할당

rvalue

값 호출



산술연산자

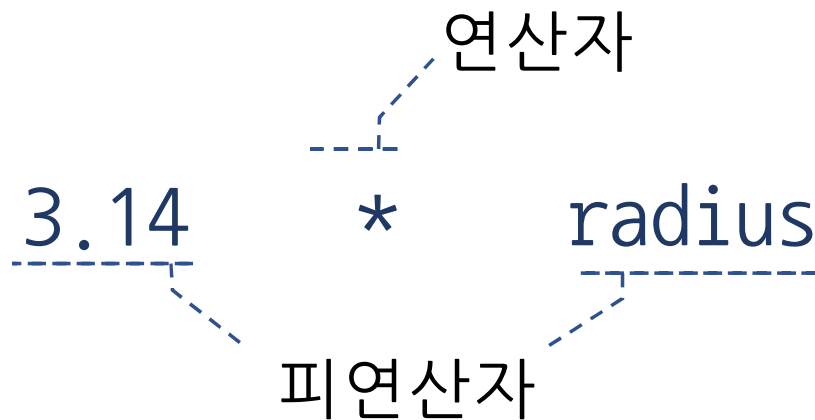
Python Programming

산술연산자의 정의

👉 피연산자(operand)에 대해 지정된 산술 연산을 지시하는 기호

→ 단항 연산자(unary operator)

→ 이항 연산자(binary operator)



특수 산술연산자

👉 프로그래밍 언어에서만 사용되는 연산 또는 부호

→ 정수 나눗셈 연산자(//)

→ 모듈러 연산자(%)

5 / 2

25 % 7

5 // 2

8.4 % 0.9

number % 2

연산자 우선순위

👉 표현식에 여러 연산자의 연산 순서를 결정

- ① 괄호 내부의 수식
- ② 지수(**) 연산자
- ③ 곱셈, 실수 나눗셈, 정수 나눗셈, 나머지 연산자
 - 왼쪽에서 오른쪽 순서로 적용
- ④ 덧셈, 뺄셈 연산자
 - 왼쪽에서 오른쪽 순서로 적용
- ⑤ 할당 연산자

```
avg = 1 // 3 * 3.14 * 20 ** 2 * (30 + 20 % 10)
```


파이썬 내장 함수

👉 파이썬 인터프리터에서 기본적으로 지원하는 함수

→ 별도의 모듈이나 패키지 없이 사용 가능

```
max(2, 3, 4)
```

```
round(3.4)
```

```
min(2, 3, 4)
```

```
abs(-3)
```

```
round(3.141592)
```

```
pow(2, 3)
```

5강

파이썬 프로그래밍 기초

제어 구조



>>> 컴퓨터과학과 정재화 교수

>>> 01

제어 구조

Python Programming

제품 생산 공정



구조적 프로그래밍

- 👉 goto 문을 사용하지 않고 프로그램을 3가지 제어 구조만으로 구성하는 프로그래밍 패러다임
 - 순차(sequence) 구조
 - 선택(selection) 구조
 - 반복(iteration) 구조
- 👉 프로그램 실행 흐름을 간결하고 작은 규모로 조직화하기 쉬움



구조적 프로그래밍

Edgar Dijkstra: Go To Statement Considered Harmful

Go To Statement Considered Harmful

Key Words and Phrases: go to statement, jump instruction, branch instruction, conditional clause, alternative clause, repetitive clause, program intelligibility, program sequencing
CR Categories: 4.22, 5.23, 5.24

EDITOR:

For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce. More recently I discovered why the use of the go to statement has such disastrous effects, and I became convinced that the go to statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

My first remark is that, although the programmer's activity ends when he has constructed a correct program, the process taking place under control of his program is the true subject matter of his activity, for it is this process that has to accomplish the desired effect; it is this process that in its dynamic behavior has to satisfy the desired specifications. Yet, once the program has been made, the "making" of the corresponding process is delegated to the machine.

My second remark is that our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do (as wise programmers aware of our limitations) our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.

Let us now consider how we can characterize the progress of a process. (You may think about this question in a very concrete manner: suppose that a process, considered as a time succession of actions, is stopped after an arbitrary action, what data do we have to fix in order that we can redo the process until the very same point?) If the program text is a pure concatenation of, say, assignment statements (for the purpose of this discussion regarded as the descriptions of single actions) it is sufficient to point in the program text to a point between two successive action descriptions. (In the absence of go to statements I can permit myself the

dynamic progress is only characterized when we also give to which call of the procedure we refer. With the inclusion of procedures we can characterize the progress of the process via a sequence of textual indices, the length of this sequence being equal to the dynamic depth of procedure calling.

Let us now consider repetition clauses (like, while *B* repeat *A* or repeat *A* until *B*). Logically speaking, such clauses are now superfluous, because we can express repetition with the aid of recursive procedures. For reasons of realism I don't wish to exclude them: on the one hand, repetition clauses can be implemented quite comfortably with present day finite equipment; on the other hand, the reasoning pattern known as "induction" makes us well equipped to retain our intellectual grasp on the processes generated by repetition clauses. With the inclusion of the repetition clauses textual indices are no longer sufficient to describe the dynamic progress of the process. With each entry into a repetition clause, however, we can associate a so-called "dynamic index," inexorably counting the ordinal number of the corresponding current repetition. As repetition clauses (just as procedure calls) may be applied nestedly, we find that now the progress of the process can always be uniquely characterized by a (mixed) sequence of textual and/or dynamic indices.

The main point is that the values of these indices are outside programmer's control; they are generated (either by the write-up of his program or by the dynamic evolution of the process) whether he wishes or not. They provide independent coordinates in which to describe the progress of the process.

Why do we need such independent coordinates? The reason is—and this seems to be inherent to sequential processes—that we can interpret the value of a variable only with respect to the progress of the process. If we wish to count the number, *n* say, of people in an initially empty room, we can achieve this by increasing *n* by one whenever we see someone entering the room. In the in-between moment that we have observed someone entering the room but have not yet performed the subsequent increase of *n*, its value equals the number of people in the room minus one!

The unbridled use of the go to statement has an immediate consequence that it becomes terribly hard to find a meaningful set of coordinates in which to describe the process progress. Usually, people take into account as well the values of some well chosen variables, but this is out of the question because it is relative to the progress that the meaning of these values is to be understood! With the go to statement one can, of course, still describe the progress uniquely by a counter counting the number of actions

```
#include<stdio.h>
int main(void)
{
    char a,b,c,d;
    a=b=c=d='a';

a: goto h;
b: if(b<'c') goto j; else goto l;
c: a++; goto i;
d: b++; goto p;
e: if(c>'c') goto c; else goto a;
f: printf("%c%c%c%c",a,b,d,c); goto n;
g: if(a<'y') goto k; else goto j;
h: c++; goto o;
i: if (b>'c') goto q; else goto g;
j: d++; goto m;
k: goto c;
l: d--; goto u;
m: if(d=='d') goto t; else goto b;
n: return 0;
o: if(a=='r') goto r; else goto e;
p: if(b>'d') goto i; else goto d;
q: a--; goto s;
r: if(c>'l') goto b; else goto h;
s: if(a<'s') goto h; else goto q;
t: printf("%c%c%c%c",c,b,a,d); goto d;
u: if(d<'b') goto f; else goto b;
v: goto a;
}
```

순차 구조

👉 실행의 흐름을 주어지는 명령의 위치적 흐름에 따라 수행하는 구조

→ 명령 라인 위에서 아래로 흐르는 가장 직관적인 구조



선택 구조

☞ 특정 영역 내의 명령문에 대한 실행 여부를 판단에 따라 결정하는 구조

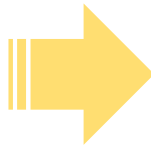
→ 실행 여부는 조건에 따라 결정



반복 구조

👉 특정 영역의 명령문을 여러 번 재실행하는 구조

→ 반복 횟수를 조건에 따라 결정



>>> 02

순차 구조

Python Programming

순차 구조

👉 실행의 흐름을 주어지는 명령의 위치적 흐름에 따라 수행하는 구조

→ 명령 라인 위에서 아래로 흐르는 가장 직관적인 구조

→ 일단 첫 단계를 시작하면 마지막 단계까지 수행





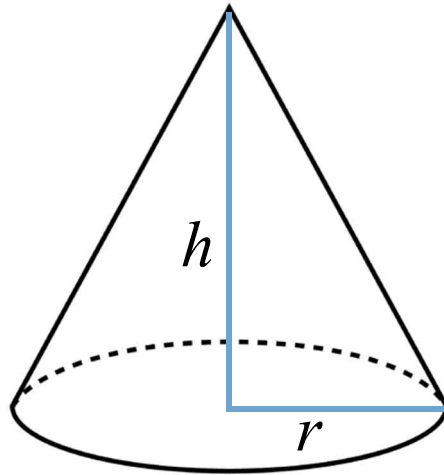
원뿔 계산 프로그램 개선



사용자로부터 반지름과 높이를 입력받고 계산

$$\text{부피} = \frac{1}{3}\pi r^2 h$$

$$\text{겉넓이} = \pi r^2 + \pi r h$$





사용자 입력



- input: 사용자로부터 데이터를 입력받는 함수
 - 입력 데이터를 문자 데이터 타입으로 반환
 - 함수의 파라미터는 입력 안내문의 목적으로 사용

```
radius = 30
```

```
#반지름 사용자 입력  
radius = input("반지름을 입력하세요")
```



원뿔 계산 프로그램 개선



사용자로부터 반지름과 높이를 입력받고 계산

$$\text{부피} = \frac{1}{3}\pi r^2 h$$

$$\text{겉넓이} = \pi r^2 + \pi r h$$

```
#반지름 사용자 입력
radius = input("반지름을 입력하세요:")
#높이 사용자 입력
height = input("높이를 입력하세요:")
#부피 계산
volume = 1/3 * 3.14 * radius ** 2 * height
print(volume)
```

데이터 타입 변환

데이터 타입을 다른 타입으로 전환

- 문자열 타입으로 변환: str 함수
- 정수 타입으로 변환: int 함수
- 소수 타입으로 변환: float 함수

데이터 타입 변환

데이터 타입을 다른 타입으로 전환

```
radius = 30
```

```
radius = input("반지름을 입력하세요")  
radius = int(radius)
```

```
radius = int("30")
```

```
radius = int(input("반지름을 입력하세요"))
```




원뿔 계산 프로그램 개선



사용자로부터 반지름과 높이를 입력받고 계산

```
1 #반지름 사용자 입력
2 rad = int(input("반지름을 입력하세요:"))
3 #높이 사용자 입력
4 hei = int(input("높이를 입력하세요:"))
5 #부피&겉넓이 계산
6 vol = 1/3 * 3.14 * rad ** 2 * hei
7 suf = 3.14 * rad ** 2 + 3.14 * rad * hei
8 print(vol)
9 print(suf)
```



print 함수의 확장



여러 개의 데이터를 단일 함수로 출력 가능

→ 콤마(,)로 파라미터를 구분하여 입력

→ 데이터 사이에 공백(기본값)이 자동으로 추가

→ sep 옵션을 변경하여 공백 변경 가능



print 함수의 확장



여러 개의 데이터를 단일 함수로 출력 가능

```
print(rad, suf)
```

```
print("원뿔의 부피는", 30, "입니다.")
```

```
print("원뿔의 부피는", vol, "입니다.", sep=",")
```

```
print("원뿔의 부피는 ", vol, "입니다.", sep=" ")
```



원뿔 계산 프로그램 개선(최종)



print 함수는 여러 개의 데이터를 출력 가능

```
1 #반지름 사용자 입력
2 rad = int(input("반지름을 입력하세요:"))
3 #높이 사용자 입력
4 hei = int(input("높이를 입력하세요:"))
5 #부피&겉넓이 계산
6 vol = 1/3 * 3.14 * rad ** 2 * hei
7 suf = 3.14 * rad ** 2 + 3.14 * rad * hei
8 print("원뿔의 부피는" ,vol, "입니다.")
9 print("원뿔의 겉넓이는" ,suf, "입니다")
```

6강

파이썬 프로그래밍 기초

선택 구조



>>> 컴퓨터과학과 정재화 교수

선택 구조의 개념

☞ 특정 영역 내의 명령문에 대한 실행 여부를 판단에 따라 결정하는 구조

→ 실행 여부는 조건에 따라 결정





선택 구조의 구문형식



구문형식

if 불리언식:

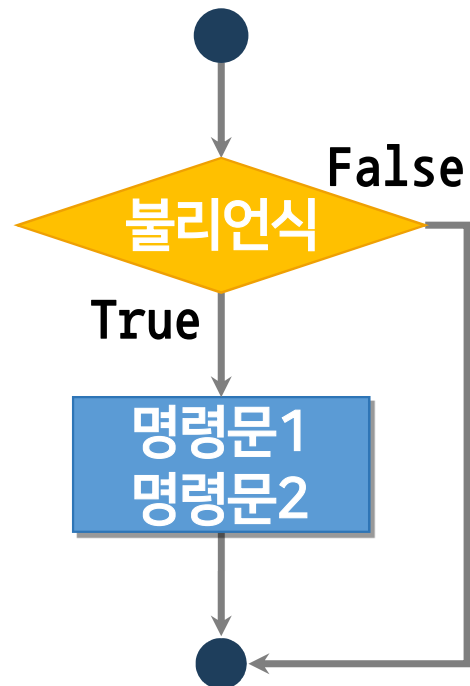
↳ 명령문1

↳ 명령문2

- 들여쓰기는 코드 블록을 표현
- 특정 동작을 수행하는 한 라인 이상의 명령문의 집합
- 스페이스 4칸 권고(PEP-8)

선택 구조의 실행흐름

실행흐름



불리언식

👉 비교연산자를 사용하여 결과가 불리언 타입으로 생성되는 표현식

연산자	수학적 표현	의미
<	<	작다.
<=	≤	작거나 같다.
>	>	크다.
>=	≥	크거나 같다.
==	=	같다.
!=	≠	같지 않다.

불리언 타입

👉 논리값인 참(True)과 거짓(False)의 값만 표현할 수 있는 데이터 타입

→ True 또는 False 예약어를 사용하여 표현

→ 비교 연산자를 사용한 표현식의 결과로 생성

```
3 > 6
```

```
light_on = True
```

```
suf == vol
```

```
isStop = False
```

원뿔 계산 프로그램 문제

👉 사용자가 반지름 값에 음수를 입력하면?

```
1 #반지름 사용자 입력
2 rad = int(input("반지름을 입력하세요:"))
3 #높이 사용자 입력
4 hei = int(input("높이를 입력하세요:"))
5 #부피&겉넓이 계산
6 vol = 1/3 * 3.14 * rad ** 2 * hei
7 suf = 3.14 * rad ** 2 + 3.14 * rad * hei
8 print("원뿔의 부피는" ,vol, "입니다.")
9 print("원뿔의 겉넓이는" ,suf, "입니다")
```

원뿔 계산 프로그램 문제

👉 사용자가 반지름 값에 음수를 입력하면 결과를 생성하지 않도록 수정

```
1 #반지름 사용자 입력
2 rad = int(input("반지름을 입력하세요:"))
3 if (rad > 0):
4     #높이 사용자 입력
5     hei = int(input("높이를 입력하세요:"))
6     #부피&겉넓이 계산
7     vol = 1/3 * 3.14 * rad ** 2 * hei
8     suf = 3.14 * rad ** 2 + 3.14 * rad * hei
9     print("원뿔의 부피는" ,vol, "입니다.")
10    print("원뿔의 겉넓이는" ,suf, "입니다")
```