
왕초보를 위한 Python: 쉽게 풀어 쓴 기초 문 법과 실습

전뇌해커

Contents

| | |
|-------------------------|----|
| 문서정보 | 1 |
| 프로그램 코드의 추가와 삭제 | 2 |
| 0. 머리말 | 3 |
| 예제 코드 | 3 |
| 2020년 | 3 |
| 2013년 | 4 |
| 0.1 주요 변경 이력 | 6 |
| 옛날 모습 | 10 |
| 1. 파이썬 시작하기 | 11 |
| 1.1 파이썬 맛보기 | 12 |
| 웹브라우저에서 파이썬을 사용하기 | 12 |
| 덧셈 | 12 |
| 뺄셈 | 14 |
| 곱셈 | 14 |
| 나눗셈 | 14 |
| 나머지 | 15 |
| 몫과 나머지를 한 번에 구하기 | 15 |
| 1.1.1 퀴즈: 사칙 연산 | 16 |
| 참고 | 16 |
| 1.2 변수 | 17 |
| 정수값을 가리키는 변수 | 17 |

| | |
|-------------------------------------|-----------|
| 변수를 이용해 숫자를 계산하기 | 18 |
| 변수의 값을 계산해 같은 변수에 다시 대입하기 | 18 |
| 문자열을 가리키는 변수 | 19 |
| 1.2.1 연습 문제: 파일 크기 계산 | 20 |
| 문제 | 20 |
| 답 | 20 |
| 1.3 리스트 (list) | 21 |
| len() | 21 |
| remove() | 22 |
| 1.4 인터프리터와 컴파일러 | 24 |
| 컴퓨터가 알아듣는 말 | 24 |
| 인터프리터와 컴파일러 | 25 |
| 1.5 파이썬 설치와 실행 | 26 |
| 파이썬 설치 | 26 |
| 일반적인 설치 | 26 |
| 과학, 수학, 데이터 분석 목적의 배포판 설치 | 26 |
| 마이크로소프트 스토어 | 27 |
| 맥 또는 리눅스 환경 | 28 |
| 파이썬 셸 실행하기 | 30 |
| 1.6 파이썬 인터프리터 | 33 |
| 인터프리터를 대화식으로 사용하기 | 33 |
| 인터프리터로 프로그램 파일을 실행하기 | 33 |
| 거북이 | 38 |
| 1.6.1 연습 문제: 제곱 | 39 |
| 문제 | 39 |
| 예 1 | 39 |
| 예 2 | 39 |
| 답 | 39 |

| | |
|--|-----------|
| 2. 제어 구조 | 40 |
| 2.1 while 을 사용하는 반복문 | 41 |
| while 문 | 41 |
| 변수값 증가 | 42 |
| while 문 수행 과정 | 43 |
| while 문 실습 | 44 |
| 2.1.1 연습 문제: 입력받은 숫자만큼 반복하기 (while) | 46 |
| 문제 | 46 |
| 예 1 | 46 |
| 예 2 | 46 |
| 답 | 47 |
| 2.1.2 연습 문제: 제곱표 (while) | 48 |
| 문제 | 48 |
| 예 1 | 48 |
| 예 2 | 48 |
| 답 | 49 |
| 2.1.3 연습 문제: 암체공 | 50 |
| 문제 | 50 |
| 답 | 51 |
| 2.1.4 연습 문제: 코드를 보고 실행 결과 맞히기 | 52 |
| 문제 | 52 |
| 풀이 | 52 |
| 2.2 조건문 (if-elif-else) | 53 |
| 파이썬의 if 와 else | 54 |
| elif | 54 |
| == 연산자 | 55 |
| 나머지 계산을 이용하는 if 문 | 55 |
| 조건에 따라 반복문 중단하기 | 55 |

| | |
|--------------------------------------|-----------|
| 2.2.1 연습 문제: 숫자 읽기 (1~3) | 57 |
| 문제 | 57 |
| 예 1 | 57 |
| 예 2 | 57 |
| 답 | 58 |
| 2.2.2 연습 문제: 나이에 따른 세대 구분 (1) | 59 |
| 문제 | 59 |
| 예 | 59 |
| 답 | 60 |
| 2.2.3 연습 문제: 단위 기호 | 61 |
| 실행 | 61 |
| 문제 | 62 |
| 풀이 | 62 |
| 2.2.4 연습 문제: 양수만 덧셈하기 | 63 |
| 문제 | 63 |
| 예 | 63 |
| 예 2 | 63 |
| 답 | 64 |
| 2.2.5 연습 문제: 윤년 판별하기 | 65 |
| 문제 | 65 |
| 답 | 67 |
| 참고 | 67 |
| 2.2.6 and/or 연산자 | 68 |
| if 문에 and/or 를 사용 | 68 |
| if 문 없이 and/or 만 사용 | 69 |
| and/or 연산 순서 | 69 |
| 2.2.7 연습 문제: 나이에 따른 세대 구분 (2) | 71 |
| 문제 | 71 |
| 예 | 71 |

| | |
|--|-----------|
| 풀이 | 72 |
| 2.3 for 를 사용하는 반복문 | 74 |
| for 문 | 74 |
| range() | 75 |
| 거북이 | 76 |
| 2.3.1 연습 문제: 입력받은 숫자만큼 반복하기 (for) | 77 |
| 문제 | 77 |
| 예 1 | 77 |
| 예 2 | 77 |
| 답 | 78 |
| 2.3.2 연습 문제: 제곱표 (for) | 79 |
| 문제 | 79 |
| 예 1 | 79 |
| 예 2 | 79 |
| 답 | 80 |
| 2.3.3 연습 문제: 화학 실험실 | 81 |
| 문제 | 81 |
| 답 | 82 |
| 2.4 match-case 문 | 84 |
| 홀수, 짝수 판별 | 84 |
| 피즈버즈 | 85 |
| 참고 | 86 |
| 2.5 for-else 와 while-else | 87 |
| for-else | 87 |
| while-else | 88 |
| 3. 함수 | 89 |
| 3.1 함수 | 90 |
| 거북이 | 92 |

| | |
|-------------------------------------|------------|
| 3.1.1 연습 문제: 자릿수를 구하는 함수 만들기 | 93 |
| 문제 | 94 |
| 예 | 94 |
| 풀이 | 94 |
| 3.1.2 연습 문제: 구구단 | 95 |
| 문제 | 95 |
| 예 | 95 |
| 힌트 | 95 |
| 풀이 | 96 |
| 2곱하기 1을 출력 | 96 |
| 2단을 출력 | 96 |
| 한 단을 계산하는 함수 | 97 |
| 구구단 전체 | 98 |
| 코드 | 98 |
| 3.2 반환 (return) 문 | 99 |
| 참과 거짓 | 100 |
| 3.2.1 연습 문제: 숫자 읽기 함수 (1~10) | 103 |
| 문제 | 103 |
| 예 | 103 |
| 답 | 103 |
| 3.2.2 연습 문제: 함수 정의하기 | 104 |
| 문제 | 104 |
| 문제 1 | 104 |
| 문제 2 | 104 |
| 답 | 105 |
| 3.2.3 연습 문제: 이자 (단리) 계산 | 106 |
| 문제 | 107 |
| 문제 1 | 107 |
| 문제 2 | 107 |

| | |
|----------------------------------|------------|
| 답 | 108 |
| 참고 | 108 |
| 3.2.4 연습 문제: 복리 계산 | 109 |
| 문제 | 110 |
| 예 1 | 110 |
| 예 2 | 111 |
| 답 | 111 |
| 참고 | 111 |
| 3.3 지역변수, 전역변수 | 112 |
| 3.5 람다 (lambda) | 115 |
| reduce() | 116 |
| filter() | 117 |
| 3.5.1 연습 문제: 놀이 공원 (1) | 119 |
| 문제 | 119 |
| 입출력 | 119 |
| 답 | 121 |
| 4. 데이터 타입 | 122 |
| 4.1 자료형 | 123 |
| 숫자 | 124 |
| 매핑 | 126 |
| 불 | 127 |
| 세트 | 127 |
| 4.1.1 연습 문제: 회문 판별 함수 만들기 | 128 |
| 문제 | 128 |
| 문제 1 | 128 |
| 문제 2 | 128 |
| 문제 3 | 129 |
| 답 | 129 |

| | |
|---|------------|
| 4.2 문자열과 리스트 | 130 |
| 문자열 | 130 |
| 리스트 | 131 |
| 문자열을 리스트로 바꾸기 | 133 |
| 숫자를 문자열로 바꾸기 | 133 |
| 문자열을 숫자로 바꾸기 | 134 |
| 리스트 원소들의 합 구하기 | 134 |
| 성적표 | 134 |
| 4.2.1 연습 문제: 각 자리 숫자의 합을 구하는 함수 (리스트를 이용) | 136 |
| 문제 | 136 |
| 예 1 | 136 |
| 예 2 | 136 |
| 힌트 | 137 |
| 답 | 137 |
| 4.2.2 연습 문제: 줄기와 잎 그림 | 138 |
| 문제 | 138 |
| 문제 1 | 138 |
| 문제 2 | 139 |
| 답 | 139 |
| 4.2.3 연습 문제: 각 자리 숫자의 합을 구하는 함수 (map() 을 이용) | 140 |
| 문제 | 140 |
| 예 1 | 140 |
| 예 2 | 140 |
| 답 | 141 |
| 4.2.4 연습 문제: 소수 구하기 | 142 |
| 문제 | 142 |
| 예 | 143 |
| 예 1 | 143 |
| 예 2 | 143 |
| 예 3 | 143 |

| | |
|------------------------------------|------------|
| 답 | 144 |
| 첫 번째 풀이 | 144 |
| 두 번째 풀이 | 144 |
| 독자분들의 풀이 | 144 |
| 프로그램 실행 시간 비교 | 145 |
| 더 읽을 거리 | 145 |
| 4.2.5 연습 문제: 진법 변환 | 146 |
| 문제 | 146 |
| 답 | 147 |
| 4.3 튜플 (tuple) | 148 |
| 4.3.1 연습 문제: 내일의 날짜 구하기 (1) | 151 |
| 문제 | 151 |
| 예 | 151 |
| 답 | 152 |
| 4.4 딕셔너리 (dict) | 153 |
| 4.4.1 연습 문제: 숫자 읽기 (0~9) | 157 |
| 문제 | 157 |
| 예 | 157 |
| 답 | 157 |
| 4.4.2 연습 문제: 한자 성어 | 158 |
| 문제 | 158 |
| 풀이 | 159 |
| 4.4.3 연습 문제: 정신 질환 | 160 |
| 문제 | 160 |
| 결과 | 160 |
| 답 | 162 |
| 4.4.4 연습 문제: 프랙털 | 163 |
| 답 | 164 |

| | |
|--|------------|
| 참고 | 164 |
| 4.5 세트 (set) | 165 |
| 4.5.1 연습 문제: 주사위 눈의 합 | 167 |
| 문제 | 168 |
| 답 | 168 |
| 4.5.2 연습 문제: 끝말 잇기 (1) | 169 |
| 예 | 169 |
| 풀이 | 171 |
| 5. 모듈 | 172 |
| 5.1 모듈이란 | 173 |
| calendar 모듈 | 174 |
| tkinter 모듈 | 174 |
| 5.2 모듈 가져오기 (import) | 176 |
| 5.2.1 연습 문제: 모듈 사용법 알아내기 | 180 |
| 답 | 181 |
| 5.2.2 연습 문제: 직각삼각형의 빗변 길이 구하기 | 182 |
| 문제 | 182 |
| 문제 1 | 182 |
| 문제 2 | 183 |
| 답 | 183 |
| 참고 | 183 |
| 5.2.3 연습 문제: calendar 와 tkinter | 184 |
| 문제 | 184 |
| 2. tkinter | 184 |
| 3. calendar 와 tkinter | 185 |
| 답 | 185 |

| | |
|--|------------|
| 5.2.4 연습 문제: 놀이 공원 (2) | 186 |
| 문제 | 186 |
| 답 | 188 |
| 5.2.5 연습 문제: 놀이 공원 (3) | 189 |
| 문제 | 189 |
| 답 | 189 |
| 5.3 여러 가지 모듈 | 190 |
| sys | 190 |
| os | 190 |
| webbrowser | 192 |
| 5.3.1 랜덤 (random) 모듈 | 193 |
| 5.3.2 연습 문제: 밴드 이름 짓기 (1) | 195 |
| 문제 | 195 |
| 예 | 195 |
| 답 | 195 |
| 5.3.3 string 과 random 모듈을 이용해 비밀번호 생성 | 196 |
| string 모듈 | 196 |
| 문자열 섞기 | 198 |
| 5.3.4 시저 (카이사르) 암호 만들기 | 199 |
| 시저 암호의 원리 | 199 |
| 파이썬으로 시저 암호 구현 | 200 |
| 참고 | 201 |
| 5.3.5. 연습 문제: 끝말 잇기 (2) | 202 |
| 예 | 202 |
| 풀이 | 204 |
| 5.3.6 연습 문제: 내일의 날짜 구하기 (2) | 205 |
| 문제 | 205 |
| 답 | 205 |

| | |
|----------------------------------|------------|
| 5.3.7 연습 문제: 원주율 구하기 | 206 |
| 원주율을 구하는 원리 | 207 |
| 거북이로 원과 정사각형 그리기 | 207 |
| 몬테카를로 방법으로 원주율을 구하는 원리 | 208 |
| random 모듈을 이용해 원주율 구하기 | 209 |
| 사각형 안에 점 찍기 | 209 |
| 점이 원 안에 있는지 판별하기 | 210 |
| 문제 | 211 |
| 풀이 | 212 |
| 참고 | 212 |
| 6. 파일 | 213 |
| 6.1 텍스트 파일 | 214 |
| 6.2 한 줄씩 다루기 | 217 |
| 6.2.1 연습 문제: 밴드 이름 짓기 (2) | 220 |
| 문제 | 220 |
| 예 | 220 |
| 답 | 221 |
| 6.2.2 연습 문제: 비밀 메시지 | 222 |
| 문제 | 223 |
| 1. 파일 읽기 | 223 |
| 2. 본문 추려내기 | 223 |
| 3. 문장부호 제거 | 224 |
| 4. 대문자로 변환 | 224 |
| 5. 비밀 메시지 출력 | 224 |
| 답 | 225 |
| 참고 | 225 |
| 6.2.2. 연습 문제: 끝말 잊기 (3) | 226 |
| 풀이 | 226 |

| | |
|--|------------|
| 6.2.3 연습 문제: 영어 퀴즈 | 227 |
| 문제 | 227 |
| 답 | 228 |
| 6.3 파일을 입맛대로 (<code>pickle</code>, <code>glob</code>, <code>os.path</code>) | 229 |
| <code>os.path</code> | 230 |
| 6.3.1 연습 문제: 애국가 | 232 |
| 문제 | 232 |
| 답 | 233 |
| 6.4 응용 예제: 음성 인식을 활용한 일본어 퀴즈 | 234 |
| 준비 | 234 |
| 아나콘다 가상환경 생성 | 234 |
| SpeechRecognition 설치 | 234 |
| PyAudio 설치 | 235 |
| 마이크 테스트 | 235 |
| 프로그램 작성 | 236 |
| 데이터 파일 | 236 |
| 파이썬 코드 | 236 |
| 참고 | 238 |
| 7. 객체지향 | 239 |
| 7.1. 클래스 (<code>class</code>) 와 인스턴스 | 240 |
| 클래스? 인스턴스? | 240 |
| 파이썬의 클래스 | 241 |
| 7.2. 변수와 메서드 | 242 |
| <code>self</code> | 244 |
| 7.3. 상속 | 245 |
| 7.3.1 메서드 상속과 재정의 | 248 |
| <code>isinstance()</code> | 250 |
| <code>issubclass()</code> | 250 |

| | |
|-------------------------------------|------------|
| 울어 () 메서드의 상속과 재정의 | 251 |
| 7.4. 객체 속의 객체 | 252 |
| 7.5. 특별한 메서드들 | 255 |
| __init__ 메서드 (초기화) | 255 |
| __del__ 메서드 (소멸자) | 256 |
| __repr__ 메서드 (프린팅) | 257 |
| __add__ 메서드 (덧셈) | 257 |
| __lt__ 메서드 (비교) | 258 |
| 8. 예외 | 260 |
| 8.1 예외처리 (try, except) | 261 |
| 더 읽을거리 | 264 |
| 8.2 연습 문제: 음성 인식 일본어 퀴즈 개선 | 265 |
| 문제 | 265 |
| 풀이 | 265 |
| 9. 테스팅과 성능 | 266 |
| 9.1 테스팅 | 267 |
| 주어진 연도가 윤년인지 아닌지를 반환하는 함수 | 268 |
| 유닛 테스트 작성과 실행 | 269 |
| 테스트 추가 | 270 |
| 모든 분기를 테스트했나? | 271 |
| 숙제 | 271 |
| 더 읽을거리 | 272 |
| 9.1.1 연습 문제: 숫자 읽기 (0~100) | 273 |
| 문제 | 273 |
| 답 | 273 |
| 영상 | 274 |

| | |
|---|------------|
| 9.2 프로그램 실행 시간 측정하기 | 275 |
| time.process_time() | 275 |
| sys.path.append() | 276 |
| importlib.import_module() | 277 |
| 실행 결과 ($N = 2^{**} 12$ 일 때) | 277 |
| 실행 결과 ($N = 2^{**} 13$ 일 때) | 278 |
| A. 부록 | 279 |
| A.1 함수의 재귀 | 280 |
| A.1.1 연습 문제: 각 자리 숫자의 합을 구하는 재귀 함수 | 284 |
| 문제 | 284 |
| 예 1 | 284 |
| 예 2 | 284 |
| 풀이 | 285 |
| A.1.2 연습 문제: 복리를 계산하는 재귀 함수 | 286 |
| 문제 | 286 |
| 문제 1 | 286 |
| 문제 2 (심화) | 287 |
| 예 3 | 288 |
| 풀이 | 288 |
| 참고 | 288 |
| A.2 연습 문제: 여러 대의 컴퓨터에 연산을 분배하기 | 289 |
| 문제 | 289 |
| 해법 | 290 |
| 공통적인 부분 | 290 |
| 해법 1 | 291 |
| 해법 2 | 292 |
| 코드 | 295 |
| A.3 진법 변환과 비트 연산 | 296 |
| 진법 변환 | 296 |

| | |
|------------------------------------|------------|
| 불값의 논리 연산 | 297 |
| 불과 정수의 관계 | 298 |
| 비트 연산 | 299 |
| 십진수의 비트 연산 | 302 |
| A.4 파이썬으로 PDF 파일 합치기 | 303 |
| PyPDF2 설치 | 303 |
| PDF 파일 준비 | 304 |
| 첫 번째 버전 | 304 |
| 두 번째 버전 | 304 |
| 세 번째 버전 | 305 |
| 사용 예 | 306 |
| 개선할 점 | 307 |
| A.5 matplotlib 으로 하트 그리기 | 308 |
| matplotlib 패키지 | 308 |
| 하트 그리기 | 308 |
| 코드 설명 | 309 |
| A.6 윈도우 CMD에서 파이썬 활용 팁 | 311 |
| 파이썬 런처 (Python Launcher) | 311 |
| Path 환경변수 | 313 |
| PATHEXT 환경변수 | 316 |
| A.7 일회용 스크립트를 재사용 가능하게 바꾸기 | 317 |
| 현재 코드 | 318 |
| 스크립트의 사용성을 개선하는 과정 | 318 |
| A.7.1 실행 가능하게 만들기 | 320 |
| 목표 | 320 |
| 절차 | 320 |
| 커밋 로그 | 320 |
| 현재 코드 | 320 |
| 스크립트 사용 예 | 321 |

| | |
|-------------------------|------------|
| A.7.2 대상을 인자로 받기 | 322 |
| 목표 | 322 |
| 절차 | 322 |
| 커밋 로그 | 322 |
| 현재 코드 | 323 |
| 스크립트 사용 예 | 323 |
| A.7.3 명령을 인자로 받기 | 324 |
| 목표 | 324 |
| 절차 | 324 |
| 현재 코드 | 324 |
| 스크립트 사용 예 | 325 |
| A.7.4 기능 추가 | 326 |
| 목표 | 326 |
| 절차 | 326 |
| 커밋 로그 | 326 |
| 현재 코드 | 326 |
| 스크립트 사용 예 | 326 |
| A.7.5 도움말 추가 | 328 |
| 목표 | 328 |
| 절차 | 328 |
| 커밋 로그 | 328 |
| 현재 코드 | 328 |
| 스크립트 사용 예 | 328 |
| Bye~ | 330 |

문서정보

Copyright© 2022 전뇌해커. All rights reserved.

이 책의 무단전재와 복제를 금합니다.

이 책은 **김효주 (mok03189@naver.com)** 님이 구매하신 전자책입니다.

(구매: <http://wikidocs.net/book/2>)

ISBN 979-11-981113-1-9



프로그램 코드의 추가와 삭제

프로그램 코드에서 초록색 배경으로 표시한 부분은 코드의 추가 또는 강조를 의미합니다.

```
def hello():
    message = "Hello"
    message = message.upper() # 추가
    print(message)
```

분홍색 배경은 삭제를 의미합니다.

```
def hello():
    message = "Hello"
    message = message.upper() # 삭제
    print(message)
```

0. 머리말

예제 코드

예제 코드는 깃허브 (Github)에 있습니다.

- <https://github.com/ychoi-kr/wikidocs-chobo-python>

2020년

제가 파이썬을 처음 접하고 이 책을 쓰기 시작한지 20년이 되었습니다. 여러 가지 프로그래밍 언어를 배우고 사용하던 중 파이썬을 알게 되면서 그 강력함과 간결함에 매료되었고, 특히 교육용 언어로서의 가능성에 주목했습니다. 제가 초등학교 때 BASIC 언어로 컴퓨터 프로그래밍을 처음 배웠던 것처럼, **초등학생도 배울 수 있는 프로그래밍 언어 교재**를 써 보자고 생각한 것이 바로 지금 보고 계신 '왕초보를 위한 파이썬'의 시작입니다.

처음에 파이썬 2.1 버전을 가지고 글을 써서 개인 홈페이지에 올리기 시작했고, 이듬해 같은 제목의 책을 출간하기도 했습니다. 나중에 위키독스로 이사를 와서 파이썬 2.7 버전을 기준으로 설명을 바꿨습니다. 파이썬 2.7은 파이썬 2의 마지막 버전이면서 파이썬 3의 구문을 지원하므로, 당시로서는 파이썬을 통해 프로그래밍 언어를 처음 배우는 사람을 위한 교재로서 최선의 선택이었습니다. 그래서 이 책에서는 그동안 파이썬 2.7을 기준으로 하되 파이썬 3의 설명도 덧붙이는 방식으로 설명했습니다.

최근 몇 년 사이 데이터 분석과 인공 지능 봄과 함께 파이썬에 대한 관심이 크게 늘었고, 어린이나 IT 분야와 직접 관련 없는 일반인을 대상으로 하는 코딩 교육의 필요성에 대한 인식도 확산되었습니다. 그 때문인지, 지금도 이 책을 읽는 분이 여전히 많이 계신 것 같습니다. 지금은 파이썬 3.8도 나와 있고, 아직 파이썬 2가 들어 있는 시스템이 있기는 하지만 프로그래밍 교육 관점에서는 파이썬 2를 크게 신경 쓰지 않아도 될 것 같습니다.

그래서, 좀 늦었지만 파이썬 3를 기준으로 설명하되 파이썬 2도 언급하는 형식으로 고쳐나가려고 합니다. 이전에도 파이썬 3를 다뤘으므로 내용이 크게 바뀌지는 않을 것 같습니다. 내용을 수정하는 동안에는 두 가지 방식의 설명이 섞여 불편하시더라도 너그럽게 양해 부탁드립니다.

용어 변경 및 설명 수정

널리 쓰이는 용어로 변경했습니다.

- 목록 → 리스트
- 사전 → 딕셔너리

또한, 설명과 예제를 쉽게 유지하면서도 좀 더 정확한 표현을 사용하려고 합니다.

연습 문제 추가

새로운 연습 문제를 추가했습니다.([주요 변경 이력 참조](#))

유튜브 영상 업로드

제 유튜브 채널 (<http://youtube.com/c/sk8erchoi>)에 영상을 새로 올렸습니다. 각 페이지에 관련 영상을 링크했고, [파이썬](#) 재생 목록에서도 보실 수 있습니다.

저자의 다른 책

이 책 외에도 제가 집필·번역한 책이 있으니 관심을 가져주시면 고맙겠습니다.

- 위키독스 책: <https://wikidocs.net/profile/info/book/4>
- 집필/번역한 책 (yes24 리스트): <http://list.yes24.com/bloglist/listList.aspx?blogid=sk8erchoi&listseqno=10608554>

2013년

2013년 현재 Python 프로그래밍 언어는 2.X 버전과 3.X 버전이 함께 사용되고 있으며, 앞으로는 3.X 버전이 주로 쓰이게 될 것입니다. Python 프로그래밍 언어를 처음 접하는 분에게는 Python 3을 배우기를 권하고 싶습니다.

지금 보고 계시는 '왕초보를 위한 Python 2.7'은 Python 2를 기준으로 작성된 것입니다. 아직 학교나 회사에서는 Python 2.X 버전을 사용하는 경우도 많을 것이기 때문에, 필요한 분들을 위해 위키독스를 통해 공개하고 있습니다. 주소는 아래와 같습니다.

- <http://wikidocs.net/book/2>

왕초보를 위한 Python: 쉽게 풀어 쓴 기초 문법과 실습

이 글을 전자책으로 읽고 계시는 분이라면 위키독스에서 구매하셨으리라 생각합니다. 부족한 책을 구입해주세요 고맙습니다. 혹시 이 책을 주변 분들에게 공유하고 싶은 마음이 드신다면, 파일을 직접 복사해주시기보다는 전자책을 구입할 수 있는 주소를 알려주시기 바랍니다. 저자에게도 도움이 되고, 좋은 온라인 북을 무료로 읽을 수 있는 위키독스의 운영을 도와주실 수 있습니다.

2013년 여름, 여러분보다 한 걸음 앞서갔을 뿐인 왕초보 최용 올림

0.1 주요 변경 이력

2023. 6.

- 연습 문제: 끝말 잇기 (1), (2), (3) 추가

2023. 5.

- 연습 문제: 원주율 구하기 추가
- 연습 문제: 한자 성어 추가

2023. 4.

- 메서드 상속과 재정의 추가
- 연습 문제: 내일의 날짜 구하기 (2) 추가

2023. 3.

- 연습 문제: 자릿수를 구하는 함수 만들기 추가
- 연습 문제: 음성 인식 일본어 퀴즈 개선 추가

2023. 2.

- for-else 와 while-else 추가
- 연습 문제: 나이에 따른 세대 구분 (1), (2) 추가
- 연습 문제: 밴드 이름 짓기 (1), (2) 추가

2022. 11.

- 시저 (카이사르) 암호 만들기 추가
- 본문과 그림을 전자책 PDF 포맷에 최적화
- 전자책 가격 인상 (3000 원 → 4000 원)

2022. 10.

- 파이썬 설치와 실행에 마이크로소프트 스토어 소개 추가
- 파이썬 인터프리터에 ‘파이썬 실행 파일 경로 확인’ 추가
- string 과 random 모듈을 이용해 비밀번호 생성 추가

2022. 9.

- and/or 연산자 추가
- match-case 문 추가 (Python 3.10 대응)

2022. 8.

- 7.5. 특별한 메서드들 수정 (`__cmp__` 삭제, `__lt__` 추가)
- 퀴즈: 사칙 연산 추가

2022. 5.

- 일회용 스크립트를 재사용 가능하게 바꾸기 추가
- 놀이 공원 연습 문제 (1), (2), (3) 추가

2022. 3.

- 연습 문제: 단위 기호 추가

2022. 2.

- 연습 문제: 프랙털 (Rule 90) 추가
- 예제 깃허브 저장소 주소 변경: <https://github.com/ychoi-kr/wikidocs-chobo-python>

2021. 7.

- 응용 예제: 음성 인식을 활용한 일본어 퀴즈 추가

2021. 6.

- 1 장의 절 순서 변경
- 강의 영상 주소 QR 코드 이미지 추가
- 프로그램 실행 시간 측정하기 추가
- 일러스트 추가 (Zdenek Sasek)

2021. 5.

- 영어 퀴즈 연습 문제 추가

2021. 4.

- 파일 크기 계산 연습 문제 추가
- 윈도우 CMD에서 파이썬 활용 팁 추가
- 부록으로 이동
 - 함수의 재귀 본문과 연습 문제
 - 연습 문제: 여러 대의 컴퓨터에 연산을 분배하기
 - 파이썬으로 PDF 파일 합치기
 - matplotlib으로 하트 그리기
- 모듈 사용법 알아내기 연습 문제 추가
- 숫자 읽기 (아라비아 숫자에 해당하는 한글을 출력) 연습 문제 추가
 - 조건문
 - 함수
 - 딕셔너리
 - 테스팅
- 세트를 별도 페이지로 분리
- 주사위 눈의 합 연습 문제 추가

2021. 3.

- 애국가 연습 문제 추가
- 정신 질환 연습 문제 추가
- 회문 판별 함수 만들기 연습 문제 추가
- 직각삼각형의 빗변 길이 구하기 연습 문제 추가
- 함수 정의하기 연습 문제 추가
- 진법 변환과 비트 연산 추가
- 진법 변환 연습 문제 추가
- 소수 구하기 연습 문제를 2 장에서 4 장으로 이동

2021. 2.

- 내일의 날짜 구하기 연습 문제 추가

- 코드를 보고 실행 결과 맞히기 연습 문제 추가
- 소수 구하기 연습 문제의 두 번째 풀이 추가
- calendar 와 tkinter 연습 문제 추가
- 비밀 메시지 연습 문제 추가

2021. 1.

- 복리 이자를 재귀적으로 계산하는 함수 연습 문제 추가
- 단리 이자 계산, 복리 이자 계산 연습 문제 추가
- 줄기와 잎 그림 연습 문제 추가

2020. 12.

- 각 자리 숫자의 합을 구하는 함수 (리스트를 이용) 연습 문제 추가
- 테스팅 추가
- 입력받은 숫자만큼 반복하기 (for) 연습 문제 추가
- 제곱표 (for) 연습 문제 추가
- 파이썬으로 PDF 파일 합치기 추가

2020. 11.

- 예제 깃허브 저장소 생성
- 양수만 덧셈하기 연습 문제 추가
- 입력받은 숫자만큼 반복하기 (while) 연습 문제 추가
- 제곱 연습 문제 추가

2020. 10.

- 화학 실험실 연습 문제 추가

2020. 9.

- 소수 구하기 연습 문제 추가

2020. 7.

- 윤년 판별하기 연습 문제 추가
- 제곱표 연습 문제 추가
- 각 자리 숫자의 합을 구하는 재귀 함수 연습 문제 추가

- [암체공 연습 문제 추가](#)

2011.

- [위키독스](#)로 이전

2003. 10.

- 카페 24로 이전

2001.

- 강좌 시작 (네띠앙)

옛날 모습

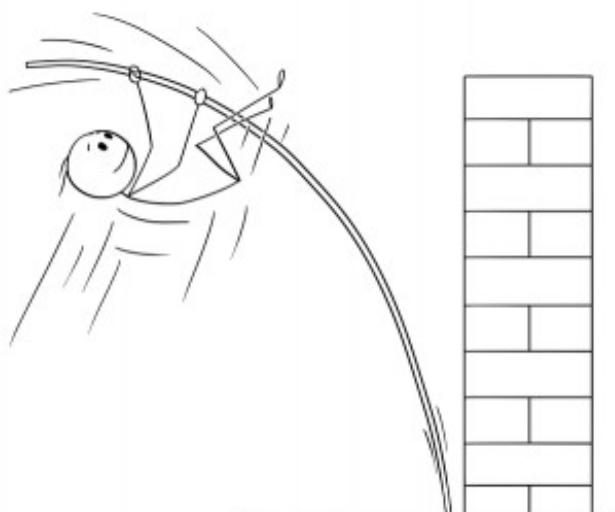
- [2004. 3. 28.\(카페 24\)](#)
- [2002. 2. 1.\(네띠앙\)](#)

1. 파이썬 시작하기

이 강좌는 프로그래밍을 처음 하는 분들을 위해서 파이썬을 통해 프로그래밍의 기초를 차근차근 익힐 수 있도록 진행하려고 합니다. 우리 함께 파이썬 탐험을 떠나 볼까요 ~

이 장에서 배우는 것:

- 파이썬 맛보기
- 변수
- 리스트 (list)
- 인터프리터와 컴파일러
- 파이썬 설치
- 파이썬 인터프리터



1.1 파이썬 맛보기

- 강의 영상: <https://youtu.be/ymSAvh-hntA/>

프로그래밍은 재미있습니다. 진짜로 재미있습니다.

그 재미를 알려면 프로그래밍을 직접 해봐야하는데, 사실 프로그래밍을 시작하는 것이 쉽지가 않습니다. 그래서 여러분과 함께 배우기 쉬운 파이썬이라는 언어를 함께 공부해보려고 합니다.

프로그래밍이 뭐냐고요? 사람이 컴퓨터에게 일을 시키려면 컴퓨터가 알아듣는 말로 일을 시켜야만 한답니다. 컴퓨터가 알아듣는 말로 그런 소프트웨어를 만드는 것이 바로 프로그래밍입니다. 그리고 컴퓨터가 알아듣는 말을 프로그래밍 언어라고 하지요.

우리가 앞으로 배울 파이썬 언어는, 배우기 쉬우면서도 프로그램을 빨리 개발할 수 있고, 기능도 뛰어나답니다.

웹브라우저에서 파이썬을 사용하기

파이썬을 설치하지 않고도 사용해 볼 수 있습니다.

파이썬 공식 홈페이지의 첫 화면에 있는 노란색 **Launch Interactive Shell** 아이콘을 클릭해보세요. 아래 그림과 같은 파이썬 셀이 뜰 거예요.

- 파이썬 공식 홈페이지: <http://python.org>

또는, 파이썬 튜터 (<http://pythontutor.com/>) 나 ideone(<https://ideone.com/>) 같은 웹사이트에서도 파이썬 코드를 작성하고 실행해 볼 수 있어요! (파이썬 튜터는 2 장의 while 문을 설명하는 [영상](#)에서 소개)

덧셈

그럼 뭐부터 해볼까요. 쓱싹쓱싹 (손 비비는 소리)... 더하기 빼기가 좋겠군요.

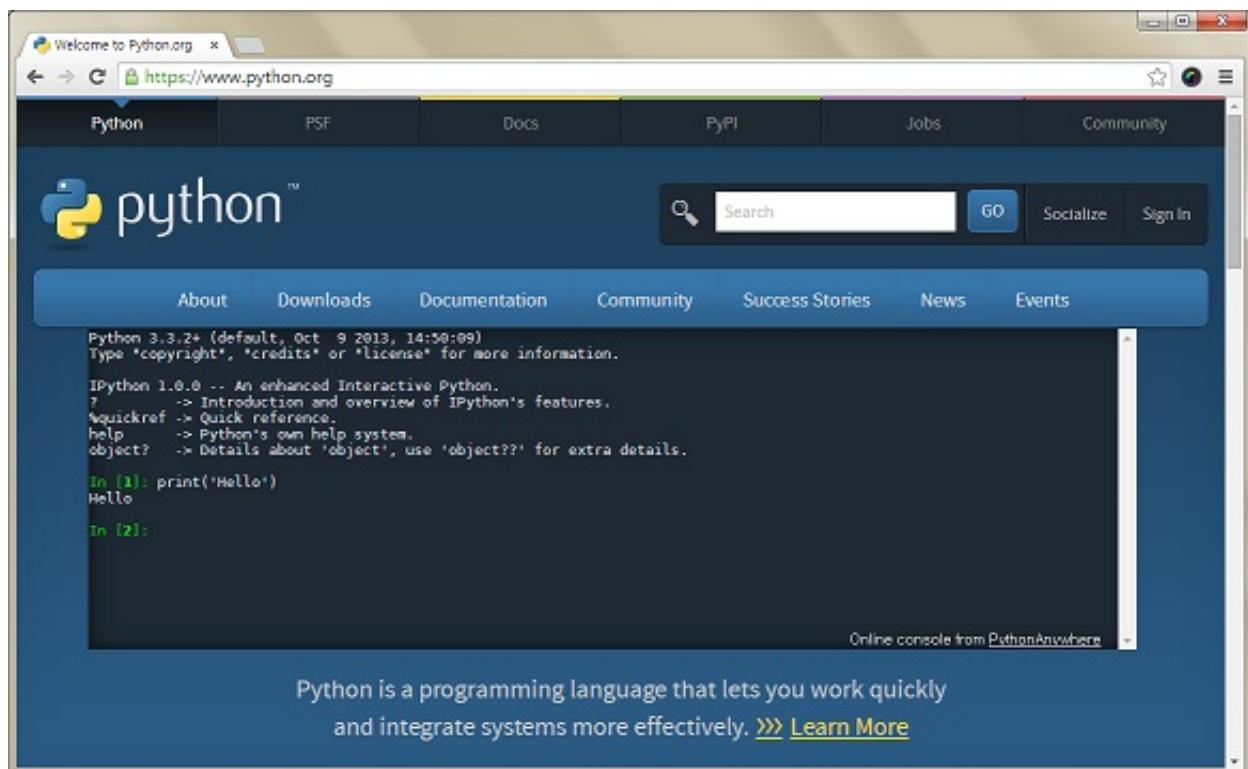


Figure 1: 파이썬 공식 홈페이지의 파이썬 셸

```
>>> 1 + 2
```

위와 같이 쓰고 Enter 키를 눌러보세요. 답이 나옵니까? 무지 쉽군요.

뺄셈

그럼 빼기도 해볼까요?

```
>>> 50 - 4  
46
```

역시 생각대로 잘 되는군요. 윈도우 첨 배울 때보다도 훨씬 쉽답니다. 하하하.

컴퓨터 켜고 끄는 법보다 더 쉽지 않습니까???

곱셈

곱하기 나누기까지만 하고 오늘은 쉴까요? 원래 첫 수업은 가볍게 시작하는 거니까요.

그럼 이번엔 좀 더 어려운 숫자를 이용해서 곱셈을 시켜 봅시다.

```
>>> 12345678 * 3  
37037034
```

곱하기도 성공! 알고 계시겠지만 컴퓨터에서는 *가 곱하기를 뜻한답니다. 나누기는요?

나눗셈

그렇습니다. /가 나누기지요. 이 정도는 기본이지요? 그럼 나눗셈도 해볼까요? (나눗셈은 파이썬 3 와 파이썬 2 의 결과가 달라요)

Python 3:

```
>>> 5000 / 3  
1666.666666666667  
>>> 5000 // 3  
1666
```

Python 2:

```
>>> 5000 / 3  
1666
```

나머지

또, 나머지를 구할 땐 %라는 연산자를 쓰면 됩니다. 예를 들어서, 50 을 8 로 나눈 나머지를 구해 볼까요?

```
>>> 50 % 8  
2
```

\$ $50 = 8 * 6 + 2$, 그러니까 50 을 8 로 나눈 몫은 6이고 나머지는 2니까 50 % 8의 결과는 2가 됩니다.

훌륭하군요. 나눗셈도 멋지게 해치우는 우리의 파이썬!

몫과 나머지를 한 번에 구하기

다음과 같이 **divmod()** 함수를 써서 몫과 나머지를 한 번에 계산할 수도 있답니다.

```
>>> divmod(50, 8)  
(6, 2)
```

함수 (function) 는 이와 같이 여러 단계의 계산을 미리 정해둔 것으로 생각할 수 있습니다. [3 장](#)에서 함수를 직접 만드는 법을 알아봅니다.

그럼 오늘의 수업은 이만 줄이고 친구 만나러 가볼까요?

1.1.1 퀴즈: 사칙 연산

생각해볼 만한 문제가 있어서 소개합니다.

1. 수학에서 다음을 계산한 결과는 얼마일까요?

$$8 \div 2(2 + 2)$$

2. 그렇다면, 파이썬 셀에서 다음 식을 계산한 결과는 얼마가 나올까요? 실행해보기 전에 미리 답을 예상해보세요.

```
8 // 2 * (2 + 2)
```

생각한 대로 답이 나왔나요?

참고

- 이 문제의 정답은? $8 \div 2(2+2) = \underline{[이슈텔러]}$

1.2 변수

이번 시간에 갖고 놀 것은 **변수** (variable)입니다. 변수가 무슨 뜻일까요? 9 시 뉴스에 가끔 등장하는 걸 들어보신 분이라면 대충 짐작이 가실텐지요. 수학에서도 나오고요. 플밍에서의 변수도 그와 비슷한 뜻을 갖고 있답니다.

정수값을 가리키는 변수

자, 제 책상에는 시계, 라이터, 칫솔, 펜, 일주일 째 물 마실 때 쓰고 있는 종이컵 등이 있습니다. 제 시계가 얼마일까요? 5 천 원? 아닙니다. 힌트... 제 시계는 결혼 예물이랍니다. 그냥, 제 시계 값은 10 만 원이라고 하겠습니다.

그럼 제 시계를 사고 싶은 사람이 있다고 합시다. 제 시계 값은 얼마일까요? 예, 그렇죠. 부르는 게 값입니다. 제가 백만 원 달라고 하면 백만 원짜리 시계가 되는 거지요. 하하~. 시계 하나가 10 만 원도 되고 백만 원도 되는 겁니다.

그럼 시계 값을 프로그램으로 표현해 볼까요?

```
>>> watch = 100000
```

제 시계가 십만 원이라는 것을 이렇게 써 보았습니다.

그럼, 라이터도 같은 방법으로 표현해 볼까요?

```
>>> lighter = 300
```

라이터는 길에서 도우미한테 받은 건데 300 원이라고 해봤습니다.

변수를 이용해 숫자를 계산하기

그럼, 저한테서 시계와 라이터를 사려면 얼마를 주셔야 할까요? 시계를 사려면 십만 원이 아니라 백만 원을 주셔야 한답니다. 그럼 시계 값을 올려야겠죠?

```
>>> watch = 1000000
```

아까 십만 원이던 시계가 이젠 백만 원이 되었군요. 그럼 합이 얼마인가요?

손가락 꼽으면서 계산하는 분 계시죠? 하하. 그걸 컴퓨터한테 시켜 봅시다.

```
>>> watch + lighter  
1000300
```

오우, 역시 머리 잘 돌아가는 컴퓨터가 순식간에 계산을 해줬습니다. 이렇게 변수에는 값을 여러 번 넣을 수 있습니다. 그런데 원래 있던 값을 지우고 다시 쓰기 때문에 값이 바뀌고 나면 이전에 얼마였는지 알 수가 없다는 것도 알아두세요.

변수의 값을 계산해 같은 변수에 다시 대입하기

시계를 중고로 팔아서 게임기를 사려고 합니다. 백만 원짜리 시계를 15% 할인한 가격은 얼마일까요?

```
>>> 1000000 * 0.85  
850000.0
```

(물론 $1000000 - 1000000 * 0.15$ 라든지 $1000000 * (1 - 0.15)$ 로 계산해도 되겠지만, 여기서는 위의 방법으로 설명할게요.)

마찬가지로, 어떤 변수가 가리키는 값을 15% 할인하는 것을 다음과 같은 코드로 나타낼 수 있을 것입니다.

```
>>> price = 5000  
>>> price = price * 0.85  
>>> price  
4250.0
```

위에서 `변수 = 변수 * 값` 형식의 문장은 `변수 *= 값`으로 줄여서 표현할 수도 있습니다.

```
>>> price = 5000  
>>> price *= 0.85
```

```
>>> price  
4250.0
```

문자열을 가리키는 변수

변수에는 숫자 말고 글자도 넣을 수 있답니다.

```
>>> a = 'pig'
```

요렇게 쓰면 `a`라는 변수에 '`pig`'라는 문자열(글자 여러 개)을 넣으라는 뜻입니다. 그러니까 `a`는 '`pig`'와 같다는 것이지요. 여기서 '`pig`'에 따옴표가 둘러져 있는 것을 주의해서 보셔야 합니다. 따옴표가 없으면 `pig`라는 변수로 착각을 하거든요. “`pig`는 문자열이다”라는 뜻으로 따옴표가 쓰이는 것입니다. 이번엔 `b`라는 변수에 '`dad`'라는 문자열을 넣어봅시다.

```
>>> b = 'dad'
```

정말 쉽죠? 그럼 이번에 문자열끼리 이어 붙여 볼까요?

```
>>> a + b  
pigdad
```

아까 시계랑 라이터 값을 더할 때랑 같은 방법을 썼더니 문자열들이 합쳐지지요? 재미있지 않습니까? 재미없으시다고요? 그렇다면 좀 더 재미있는 것을 해봅시다.

```
>>> a + ' ' + b  
pig dad
```

여기선 `a`라는 변수와 ' ' (공백 문자 한 개) 와 `b`라는 변수를 붙여서 '`pig dad`'라는 문자열을 만들어 준 것이지요. 이해되시지요?

오늘의 강좌는 여기까지입니다.

1.2.1 연습 문제: 파일 크기 계산

문제

파일을 다운로드할 때의 평균 속도 (average rate) 를 r 이라 하고, 다운로드하는 데 걸린 시간 (time) 을 t 라고 할 때, 다운로드한 파일의 용량은 $r \times t$ 로 계산할 수 있습니다.

다운로드 속도가 초당 800kB이고 다운로드하는 데 걸린 시간이 110 초라고 할 때, 다운로드한 파일의 크기는 몇 MB 일까요? 단, 1MB = 1000kB로 계산합니다.

답

- [ch01/download.txt](#)

1.3 리스트 (list)

- 강의 영상: <https://youtu.be/-BA3lbvggCM>

안녕하세요 ~ 여러분 ~ 오늘도 여러분과 함께 파이썬 놀이하러 왔습니다. 오늘이 제가 이 강좌를 시작한지 이틀째 되는 날입니다. 작심삼일이 되지 않도록 저에게 힘을 주세요.

저도 엊그제까지만 해도 파이썬에 대해 아무것도 몰랐답니다. 무식하면 용감하다고 저도 공부해 가면서 용감하게 글을 쓰고 있습니다. ^^;

오늘은 저희 가족 이야기를 해보겠습니다. 저희 식구는 네 명이었습니다. 어머니, 아버지, 저, 동생. 대장은 어머니, 그다음이 아버지입니다. 그러니까 저는 넘버 쓰리였던 것이었습니다... 으흐흐...

파이썬에서는 저희 가족을 이렇게 표현할 수 있답니다.

```
>>> family = ['mother', 'father', 'gentleman', 'sexy lady']
```

저희 가족에는 아버지, 어머니, 저, 동생이 있다는 것을 **리스트** (list)로 표현한 것이지요. 이해가 갑니까? 옵니까? 예, 그냥 동생이라고 했는데 실은 여동생입니다. 침 닦으십쇼. -#

len()

그럼, 컴퓨터한테 저희 가족이 몇 명인지 물어보겠습니다.

```
>>> len(family)  
4
```

len() 함수는 리스트에 **원소** (element)가 몇 개 들어 있는지 보여줍니다. 저희 가족을 `family`라는 리스트로 표현했으니까 4라고 대답을 하는 겁니다.

리스트는 말 그대로 여러 개의 자료를 묶은 것입니다. 위에서 보신 것처럼 대괄호 ([]) 랑 콤마 (,)를 써서 표현하면 됩니다.

그럼 저희 가족 중에 넘버 쓰리가 누구인지 물어볼까요?

아래와 같이 입력하고 Enter 를 살짝 눌러주세요.

```
>>> family[3]
```

답이 뭐라고 나오죠? 당근 'gentleman' 이라고 나오겠죠? 그러나! 이상하게도 파이썬은 다른 대답을 합니다.

어떤 결과가 나오는지 직접 확인해보세요.

...

확인해보셨나요? 왜 그런 답이 나왔을까요?

그 이유는 리스트의 첫번째 자리를 차고앉은 'mother' 가 1 번이 아니라 0 번이기 때문입니다. 왜 1이 아니라 0 일까요? 그건 저도 잘 모릅니다.-;

파이썬 말고 다른 프로그래밍 언어에서도 여러 개를 묶어두었을 땐 번호를 0 번부터 붙여주더군요.

그렇다면 저는 몇 번일까요? 'mother' 가 0이고, 'father' 가 1이니까 저는 2 겠군요. 맞는지 확인을 해봐야죠?

```
>>> family[2]
'gentleman'
```

제가 넘버투가 되어버린 감격스러운 순간입니다!!! 기뻐해 주십시오, 여러분 ~.

하지만 그 기쁨도 잠시, 저의 머리가 5400rpm 으로 회전하기 시작했습니다 (rpm 은 1 분에 몇 바퀴를 도는지 나타내는 단위로, 컴퓨터의 하드 디스크 회전속도가 보통 5400rpm 또는 7200rpm 입니다). 저는 넘버 투에 만족할 놈이 아니기 때문입니다.

머리를 한참 굴리던 저는 넘버 0이 될 수 있는 방법을 찾아내었습니다. 무슨 방법이냐고요?

독립입니다. 대한독립만세 ~

remove()

우선 가족에서 제 이름을 뺍니다.

```
>>> family.remove('gentleman')
```

`remove` 는 뭔가를 제거한다는 뜻을 갖고 있죠. 위의 문장은 `family`에서 '`gentleman`'이라는 놈을 없애라는 말입니다. 그럼 제가 확실히 없어졌는지 확인해보겠습니다.

```
>>> family  
['mother', 'father', 'sexy lady']
```

예, 제가 확실히 제거되었군요. 이제 새로운 세대를 구성할 차례입니다. 새로운 리스트는 `family` 말고 다른 이름을 지어주면 좋을 것 같습니다. 여러분이 새로운 이름으로 하나 만들어주세요.

1.4 인터프리터와 컴파일러

컴퓨터가 알아듣는 말

엄청난 양의 정보들을 잘도 주무르는 이 컴퓨터란 녀석은 대체 어떻게 작동하는 걸까요? 컴퓨터는 전기가 통하는지 안 통하는지의 딱 두 가지 정보를 가지고 정보를 처리한답니다. 우리가 저녁이 되면 전깃불을 켜고, 잠잘 땐 끄는 것과 마찬가지입니다. 저는 불을 켜고도 잘 자지만요.^^ 이렇게 간단한 정보를 갖고 무엇을 할 수 있을까 싶지만, 스위치가 두 개 있으면 네 가지 정보를 표현할 수 있다는 걸 생각해 보세요.

```
큰 방 불 켜고, 작은 방 불 꺼.  
큰 방 불 켜고, 작은 방도 불 켜.  
큰 방 불 끄고, 작은 방 불 켜.  
큰 방 불 끄고, 작은 방도 불 꺼.
```

스위치가 10 개 있으면 몇 가지 정보를 표현할 수 있을까요? 스위치가 두 개 있으면 2의 제곱인 4 가지 정보를 표현할 수 있었고요, 스위치가 10 개 있으면 2의 10 제곱, 그러니까 1024 가지 정보를 표현할 수 있답니다.

컴퓨터에선 불이 켜지고 꺼진 것을 1 과 0 으로 나타내고요, 스위치 하나에 해당하는 것을 **비트** (bit) 라고 말하지요. 컴퓨터에 일을 시키려면 컴퓨터가 알아들을 수 있는 말로 **지시** (instruction) 를 내려야 합니다.

이럴 때 쓰이는 프로그래밍 언어를 **저급 언어** (low-level language) 라고 합니다. 저급 언어로는 기계어와 어셈블리어가 있습니다.

그런데 사람이 1 과 0 으로 프로그래밍하는 것이 가능할까요?

“컴퓨터야, 10111101 한 다음에 01001011 하고, 만약에 10011010 이면 10101100 하고 그렇지 않으면 11010011 해다오, 알았지??”

얼마나 짜증 나겠습니까? 그런 어려움이 있기 때문에 사람이 프로그램 작성은 쉽게 할 수 있도록 요즘 많이 쓰는 C/C++, 파이썬, 자바 같은 **고급 언어** (high-level language) 가 생겨났답니다.

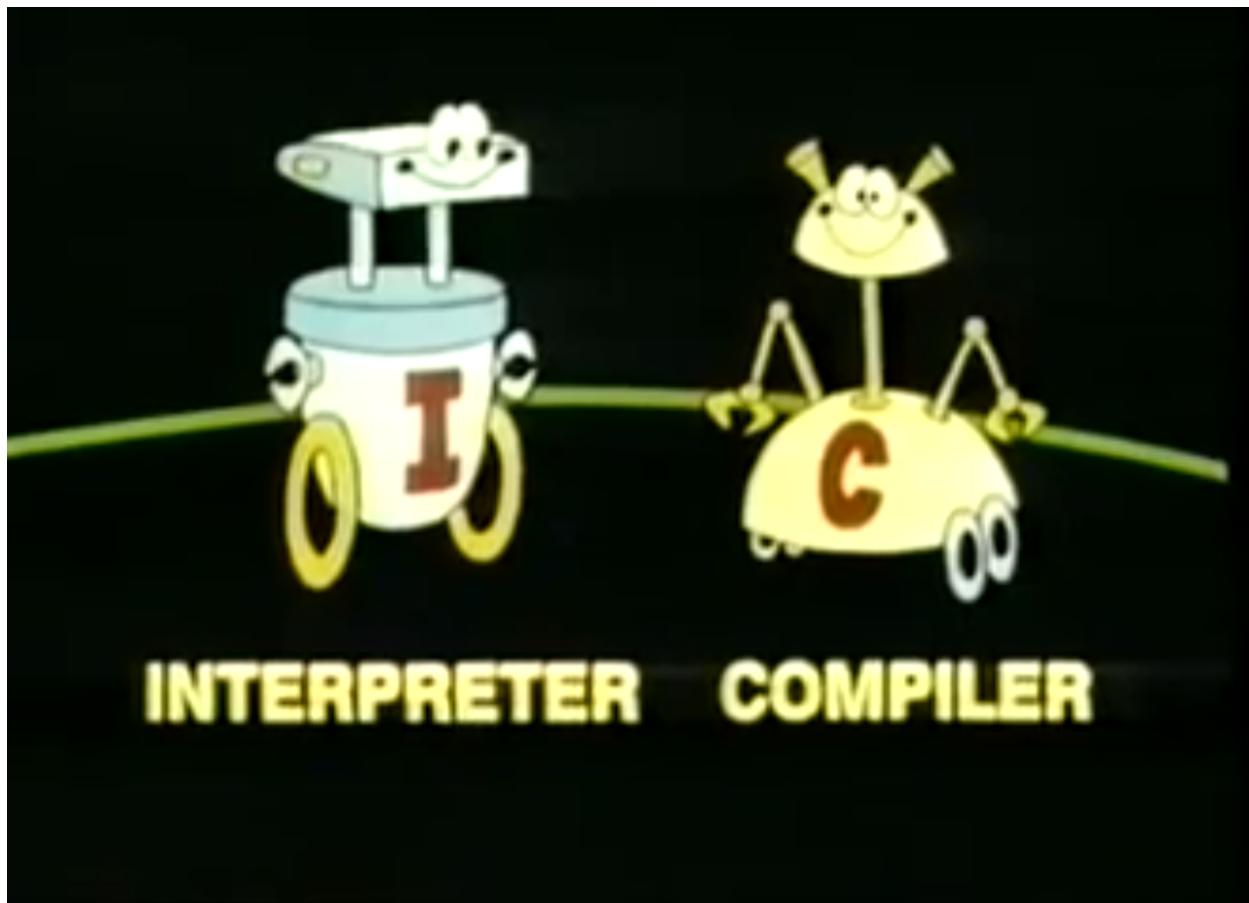
인터프리터와 컴파일러

그런데 이런 고급 언어로 프로그램을 짠 다음엔 컴퓨터가 알아들을 수 있게 번역을 해 줄 필요가 있겠죠?
그렇게 번역을 하는 방법에도 두 종류가 있답니다. (예, 압니다. 저도 종류 많은 거 딱 질색입니다...^^;)

한마디 할 때마다 동시통역해주는 방식을 **인터프리트** (interpret) 방식이라고 하고, 말하는 것을 처음부터 끝까지 듣고 나서 한꺼번에 바꿔주는 것을 **컴파일** (compile) 방식이라고 합니다.

우리가 배우는 파이썬은 어떤 방식일까요? 파이썬은 우리의 명령을 한 줄씩 해석해서 일을 하는 인터프리트 방식입니다. 사람이 파이썬 언어로 작성한 프로그램을 컴퓨터에 번역해주는 파이썬 셀이 바로 **인터프리터** (interpreter) 랍니다.

- 인터프리터와 컴파일러를 설명한 애니메이션 (우리말 자막 있음): <https://youtu.be/Dx2tSsd3aFc>



1.5 파이썬 설치와 실행

파이썬이라는 언어를 좀 더 갖고 놀아보고 싶은 마음이 드셨나요?

저는 ‘공부’ 한다는 말보다는 ‘논다’는 것이 더 마음에 드는군요. 재미있으니까요.

파이썬 설치

파이썬을 본격적으로 배우기에 앞서, 컴퓨터에 파이썬을 설치하겠습니다. 이 책에는 컴퓨터의 파일을 다루는 실습도 있으므로 파이썬을 설치할 것을 권장하지만, 설치하기 곤란한 분은 넘어가셔도 됩니다.

아래에서 소개하는 대표적인 파이썬 배포판 중 하나를 선택해 설치하시면 됩니다.

일반적인 설치

[python.org](https://www.python.org) 의 [다운로드 페이지](#)에서 파이썬 설치 프로그램을 내려받으실 수 있습니다. 최신 버전 (2023년 1월 7일 현재 [3.11.1](#)) 을 다운로드해 주세요.

파일을 받으셨으면 인스톨러를 실행해 ‘Install Now’ 버튼을 눌러 설치해 주세요.

- 파이썬 설치 동영상: <https://youtu.be/mk8lP7WLQ9E>

과학, 수학, 데이터 분석 목적의 배포판 설치

과학 계산용으로 널리 사용되는 파이썬 패키지들을 쉽게 관리할 수 있는 배포판도 있습니다. 그중 대표적인 것이 아나콘다 (Anaconda) 입니다.

이 책으로 파이썬 문법을 익힌 후 데이터 분석 및 머신러닝을 하려는 분은 처음부터 아나콘다를 설치해서 실습하셔도 됩니다.

- 아나콘다 설치 동영상: <https://youtu.be/CNGBwFcmBZM>

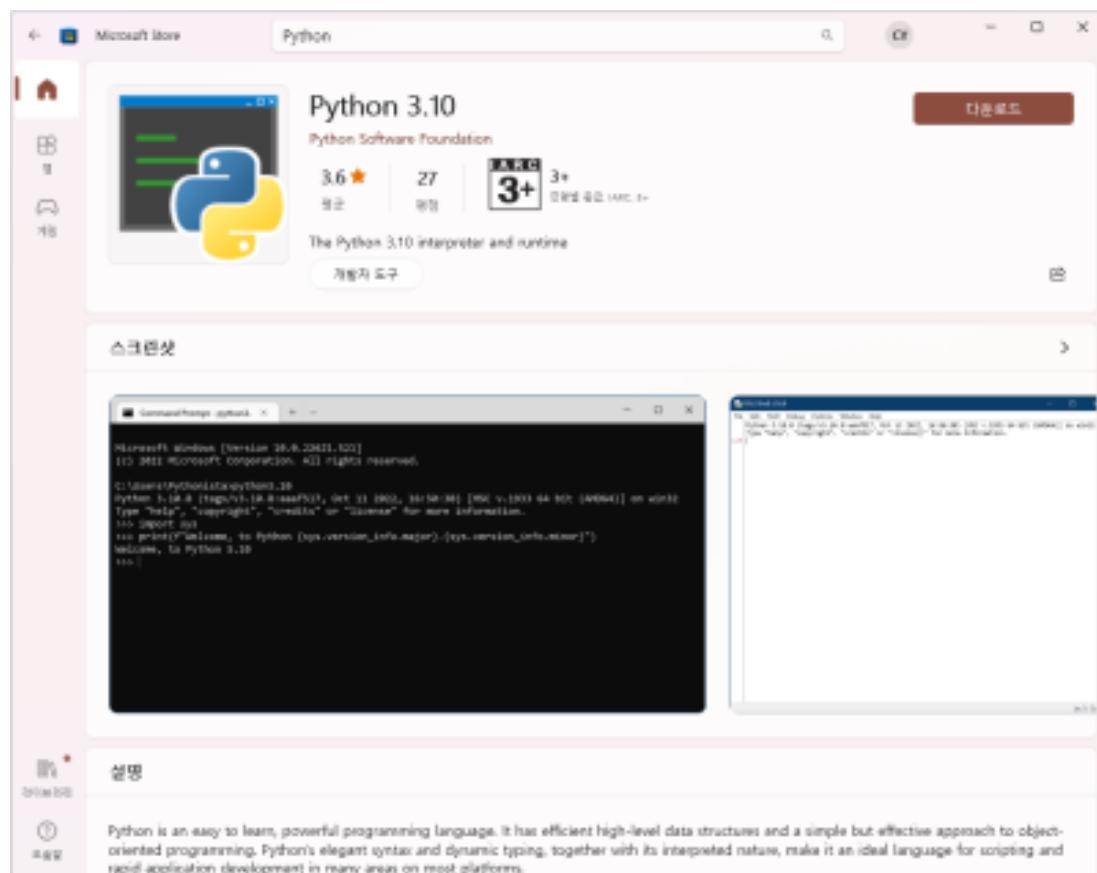
Q: 다른 파이썬 책에서 3.7 버전으로 실습하라는데, 파이썬을 지우고 새로 설치해야 하나요?

A: 파이썬 기본 문법을 배울 때는 더 높은 버전으로 실습해도 문제가 없습니다. 예를 들어, 파이썬 3.7을 기준으로 작성한 코드를 파이썬 3.9에서도 실행할 수 있습니다.

그런데 여러 가지 파이썬 패키지를 추가로 설치해서 사용하다 보면 파이썬 버전을 맞춰야 할 때가 있습니다. 그럴 때 파이썬 가상 환경 (venv 또는 conda env) 을 활용하면 가상 환경마다 파이썬 버전을 다르게 할 수 있어 편리합니다. 아나콘다 설치 동영상의 가상 환경 설명 (<https://youtu.be/CNGBwFcmBZM?t=494>) 을 참고하세요.

마이크로소프트 스토어

윈도우 운영체제에 포함된 ‘마이크로소프트 스토어 (Microsoft Store)’에서 ‘Python’을 검색해 설치할 수도 있습니다.



맥 또는 리눅스 환경

맥이나 리눅스를 쓰시는 분들은 파이썬이 이미 설치되어 있을 수도 있습니다. 터미널에서 `python` 또는 `python3`이라고 쳐보세요.

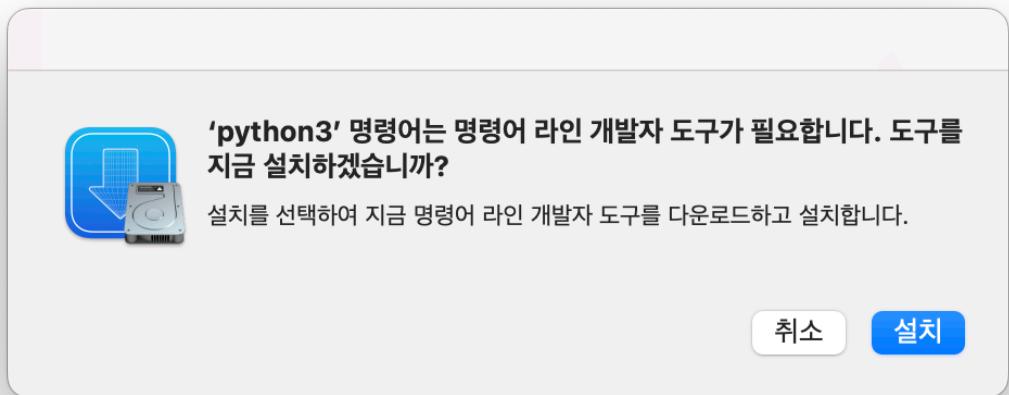
```
Mac% python
Python 2.7.10 (default, Feb  7 2017, 00:08:15)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.34)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

그런데 시스템에 기본으로 설치된 파이썬 버전이 2.7.X인 경우, 이 책에서는 파이썬 3를 기준으로 설명하므로 간혹 실행 결과가 다르게 나올 수 있습니다. 특별한 이유가 없다면 파이썬 3.8 이상을 설치하는 것이 좋습니다.

맥 OS에 파이썬 3 설치 (1)

제가 새로 산 맥북 (2022년형, M2)에 파이썬이 설치돼 있는지 보려고 터미널에서 명령을 실행했더니 아래와 같은 메시지와 함께 팝업창이 떴습니다.

```
yong@MacBookPro ~ % python --version
zsh: command not found: python
yong@MacBookPro ~ % python3 --version
xcode-select: note: no developer tools were found at '/Applications/Xcode.app',
requesting install. Choose an option in the dialog to download the command
line developer tools.
```



팝업 창의 ‘설치’ 버튼을 누르고 잠시 기다렸더니 파이썬이 설치됐습니다.

```
yong@MacBookPro ~ % python3 --version
Python 3.9.6
yong@MacBookPro ~ % python3
Python 3.9.6 (default, Oct 18 2022, 12:41:40)
[Clang 14.0.0 (clang-1400.0.29.202)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

맥 OS 에 설치 (2)

다른 방법으로, <https://www.python.org/downloads/macos/>에서 맥용 인스톨러를 다운로드해서 설치해도 됩니다.

맥에는 인텔 CPU 가 들어간 것도 있고 애플 실리콘 (M1, M2) 이 들어간 것도 있는데, 파이썬 최신 버전을 설치할 때는 어느 CPU 를 사용하는지에 관계없이 macOS 64-bit universal2 installer 를 사용해 설치하면 됩니다.

2020년 11월 ~ 2022년 5월 사이에 나온 파이썬 배포판의 맥용 인스톨러는 CPU 별로 따로 되어 있습니다. 그중에서 하나를 설치하고 싶다면, 바탕 화면 왼쪽 위의 사과 모양 아이콘을 누르고 ‘이 Mac 에 관하여’를 누른 뒤 칩을 확인합니다.

- 칩이 Apple M1 또는 Apple M2 로 표시될 때는 macOS 64-bit universal2 installer 를 다운로드합니

다.

- 인텔 칩일 때는 macOS 64-bit Intel installer 를 다운로드합니다.

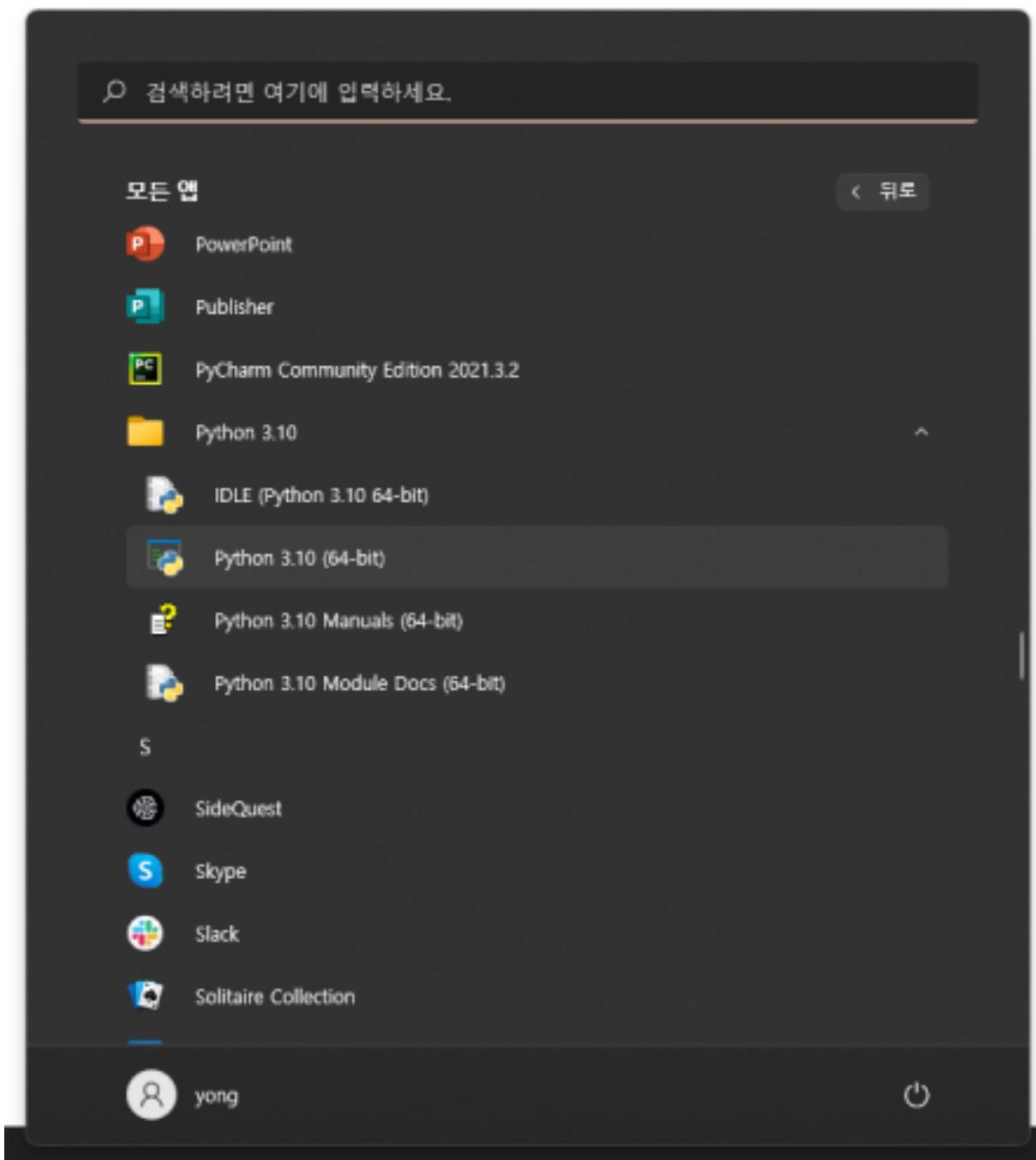
라즈베리 파이에서 파이썬 사용

다음 문서를 참고하세요. <https://wikidocs.net/3168>

파이썬 셸 실행하기

그럼 파이썬을 실행해 볼까요? 윈도우의 시작 메뉴에서 파이썬을 실행해 주세요.

예를 들어 윈도우 11에 Python 3.10(64 비트)을 설치한 경우, 윈도우 버튼 → 모든 앱 → **Python 3.10** → **Python 3.10 (64-bit)**을 선택합니다.



다음과 같이 파이썬 셀이 실행됩니다. 여러분은 앞으로 이것을 이용해서 프로그래밍을 하게 될 겁니다.

왕초보를 위한 Python: 쉽게 풀어 쓴 기초 문법과 실습



Python 3.10 (64-bit)
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>

1.6 파이썬 인터프리터

인터프리터를 대화식으로 사용하기

지금까지 우리가 사용했던, 인터프리터에 명령을 한 줄씩 입력하면 인터프리터가 그때그때 답을 돌려주는 방법을 두고 “대화식으로 인터프리터를 사용한다”라고 합니다. 당연한 말이지만 대화식은 아주 쉽고 간단한 사용방법입니다. 우리도 파이썬에 대해 아무것도 모르는 상태에서 당장 인터프리터를 계산기처럼 쓰기 시작할 수 있었지요.

대화식 사용방법은 그냥 쉽기만 한 것이 아니라, 파이썬의 문법을 배우고 연습해 보거나, 복잡한 프로그램을 작성하기 위해 작은 부분을 테스트할 때도 아주 편리하답니다.

인터프리터로 프로그램 파일을 실행하기

대화식 사용방법이 여러모로 편리하기는 하지만, 실제로 프로그램을 작성할 때는 역시 프로그램을 파일 형태로 만들어 두었다가 사용해야 겠죠? 파이썬에서는 프로그램 파일을 어떻게 만드는 걸까요?

파이썬에 포함된 IDLE 또는 여러분이 즐겨쓰는 텍스트 편집기 (메모장 같은 프로그램)에서 다음 코드를 작성해 보세요.

파이썬 3:

```
print('직각삼각형 그리기\n')
leg = int(input('변의 길이: '))

for i in range(leg):
    print('*' * (i + 1))

area = (leg ** 2) / 2
print('넓이:', area)
```

파이썬 2:

```
print('Right triangle\n')
leg = int(input('leg: '))

for i in range(leg):
    print('*' * (i + 1))

area = (leg ** 2) / 2.0
print('area:', area)
```

처음보는 것들이 많지만 나중에 배울 것들이니 겁먹지 마시고 그냥 똑같이 쓰시면 됩니다. 주의하실 점은, 반드시 들여쓰기를 지켜주셔야 한다는 것입니다. 들여쓰기를 하실 땐 공백 네 칸 씩 들여쓰기하시면 됩니다.

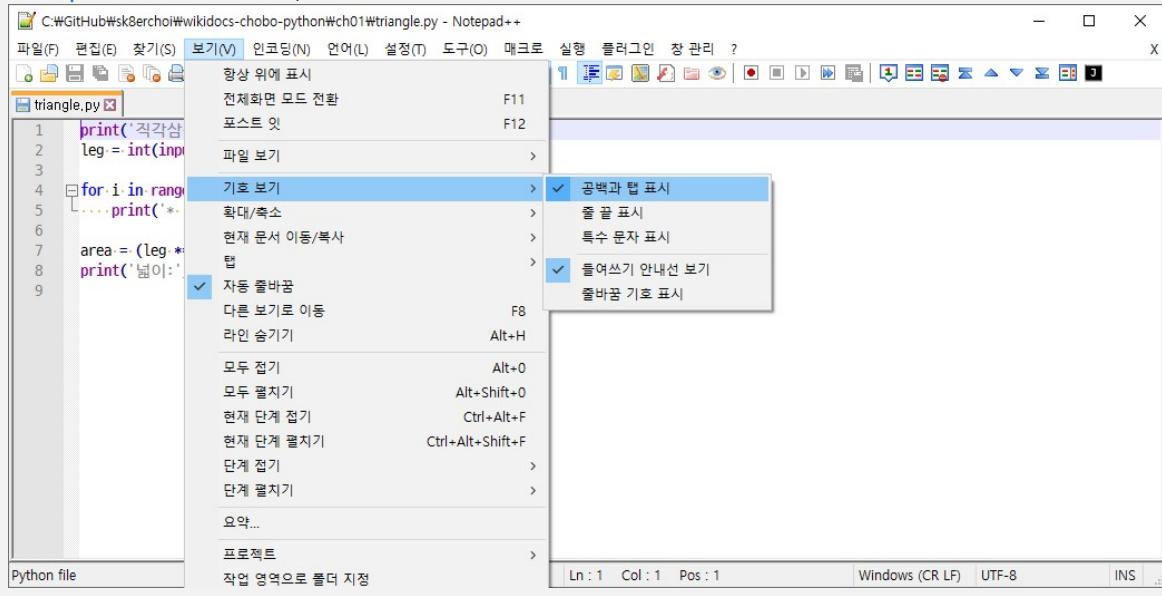
이것을 `triangle.py`라는 이름으로 저장해 주세요. 파이썬 코드 파일에는 `py`라는 확장자가 붙습니다. 저장 위치는 `\wikidocs-chobo-python\ch01`이라고 가정하고 설명합니다.

코드 작성과 실행까지 IDLE에서 할 수도 있고, 텍스트 편집기에서 작성한 후 윈도우 명령 프롬프트에서 실행할 수도 있어요.

Tip: 텍스트 편집기에 공백을 표시하기

텍스트 편집기 중에는 탭과 스페이스를 기호로 나타내는 기능이 있는 것도 있습니다.

`Notepad++`를 예로 들면, 그림과 같이 보기 → 기호 보기 → 공백과 탭 표시 메뉴를 선택하면 됩니다.



IDLE에서 실행

위 예제 코드를 IDLE에서 작성하고 실행하는 영상입니다. (파이썬 3.8) https://youtu.be/P4xAx4HFa_hU

윈도우 명령 프롬프트에서 실행

명령 프롬프트 창 열기 먼저 명령 프롬프트를 띄워보세요. 다음 두 가지 방법 중 편한 방법으로 하시면 됩니다.

- 윈도우 메뉴에서 **Windows 시스템 - 명령 프롬프트**를 선택
-  + R 을 누르고 **실행** 창에서 cmd 입력

윈도우 버전에 따라 조금씩 다르기는 하지만 아래 그림과 비슷한 창이 뜰 거예요.



Figure 1: 명령 프롬프트 창

명령어 인터페이스가 사용하기 번거롭기는 해도 좀 더 자세한 조작을 하기에 적합하지요. 다른 언어로 프로그램을 작성할 때도 종종 쓰이니까 이번 기회에 익혀두자고요.

명령 프롬프트를 실행하면 사용자의 홈 폴더에서 시작합니다. 지금 제 컴퓨터에서는 `C:\Users\yong` 이 현재 경로입니다.

파이썬 실행 파일 경로 확인 파이썬 실행 파일이 어디에 있는지는 `where`(윈도우) 나 `which`(리눅스, 맥) 명령을 써서 확인할 수 있습니다.

윈도우에서:

```
C:\Users\yong>where python  
C:\Users\yong\AppData\Local\Programs\Python\Python310\python.exe  
C:\Users\yong\AppData\Local\Microsoft\WindowsApps\python.exe
```

`where` 명령은 현재 폴더 및 PATH 환경 변수에 등록된 경로에서 파일을 찾습니다. 따라서 파이썬 설치 시 PATH 등록 옵션을 선택하지 않은 경우, `where`의 결과에 나타나지 않을 수 있습니다.

리눅스에서:

```
(base) yong@thinkpad:~$ which python  
/home/yong/anaconda3/bin/python
```

파이썬 설치 경로로 이동 파이썬이 설치된 곳으로 이동해 보겠습니다. 다음과 같이 `cd` 명령을 사용합니다.

```
cd Appdata\Local\Programs\Python
```

```
dir
```

```
cd Python310
```

cd: 지정한 디렉터리(폴더)로 이동

```
C:\Users\yong>cd Appdata\Local\Programs\Python
```

```
C:\Users\yong\AppData\Local\Programs\Python>dir
```

C 드라이브의 볼륨: Windows
볼륨 일련 번호: 60C7-D75E

dir: 현재 디렉터리의 파일 목록을 출력

C:\Users\yong\AppData\Local\Programs\Python 디렉터리

| | | |
|---------------------|---------|------------------------|
| 2022-02-25 오후 09:39 | <DIR> | . |
| 2022-04-15 오전 07:31 | <DIR> | .. |
| 2022-05-13 오후 07:03 | <DIR> | Python310 |
| | 0개 파일 | 0 바이트 |
| | 3개 디렉터리 | 212,381,741,056 바이트 남음 |

위에서 확인한 실제
디렉터리명을 입력
해주세요.

```
C:\Users\yong\AppData\Local\Programs\Python>cd Python310
```

현재 경로는 C:\Users\yong\AppData\Local\Programs\Python\Python310입니다. 이
곳에 파이썬 실행 파일이 있을 것입니다.

```
dir python*
```

```
C:\Users\yong\AppData\Local\Programs\Python\Python310>dir python*
```

C 드라이브의 볼륨: Windows
볼륨 일련 번호: 60C7-D75E

C:\Users\yong\AppData\Local\Programs\Python\Python310 디렉터리

| | | |
|---------------------|-----------|------------------------|
| 2022-01-17 오후 02:26 | 99,216 | python.exe |
| 2022-01-17 오후 02:26 | 62,352 | python3.dll |
| 2022-01-17 오후 02:26 | 4,453,776 | python310.dll |
| 2022-01-17 오후 02:26 | 97,680 | pythonw.exe |
| | 4개 파일 | 4,713,024 바이트 |
| | 0개 디렉터리 | 212,701,118,464 바이트 남음 |

python.exe 파일이
있는지 확인합니다.

파이썬으로 스크립트 실행 파이썬이 설치된 폴더를 잘 찾아오셨으면 아래와 같이 명령을 내려주세요.

```
python \wikidocs-chobo-python\ch01\triangle.py
```

이 명령은 '파이썬아, \wikidocs-chobo-python\ch01 폴더에 있는 triangle.py를 실행하자 꾸나' 라는 뜻이랍니다. 그러면 파이썬 인터프리터가 이 프로그램 전체를 번역하면서 실행시켜 줄 거예요.

직각삼각형 그리기

변의 길이: 4

```
*  
* *  
* * *  
* * * *
```

넓이: 8.0

만약 여기에서 그냥 python이라고만 입력하면 어디서 많이 본 것이 나타날 겁니다. 바로 대화식 인터프리터죠.

프로그램 파일의 실행이 잘 안 되는 분은 예제와 자신이 직접 짠 프로그램을 잘 비교해서 잘못된 부분이 있는지 잘 찾아보세요. 연습할 때 실수하는 부분이 많을수록 공부에 더 많은 도움이 된답니다.

자, 이렇게 해서 인터프리터를 사용해서 프로그램 파일을 실행시키는 방법도 알아보았습니다. 마지막으로 알려 드릴 것은, 이 파일을 그냥 더블클릭해도 실행된다는 사실... 하지만 더블 클릭이 항상 편한 방법이 아니라는 것은 금방 알게 되실 거예요.

그럼 좋은 꿈 꾸세요 ~.

- 점프 투 파이썬 - 사용자 입력과 출력도 읽어보세요

거북이

- Python 터틀 그래픽스를 사용해 여러 가지 도형 그리기: <https://youtu.be/ZEV3pGTdlxw>

1.6.1 연습 문제: 제곱

문제

사용자에게 정수를 입력받아, 그 수의 제곱을 계산해 출력하는 파이썬 스크립트를 작성하세요.

예 1

입력:

```
3
```

출력:

```
9
```

예 2

입력:

```
5
```

출력:

```
25
```

답

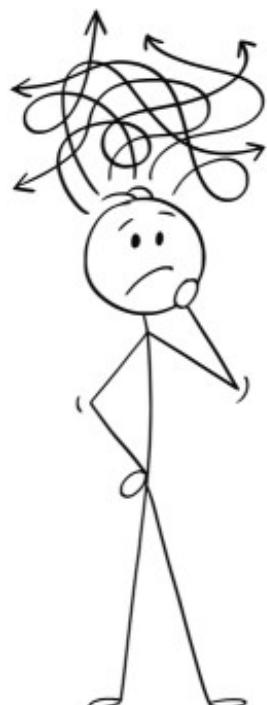
- 코드: [ch01/square.py](#)

2. 제어 구조

프로그래밍의 핵심적인 요소인 분기와 반복을 파이썬에서는 어떻게 사용하는지 살펴볼게요.

이 장에서 배우는 것:

- `while` 을 사용하는 반복문
- 조건문 (`if-elif-else`)
- `for` 를 사용하는 반복문



2.1 while 을 사용하는 반복문

- 강의 영상 https://youtu.be/j_NPpCNsfIM

오늘은 한석봉 이야기를 해보겠습니다.

큰 뜻을 품고 어머니를 떠나 숫자공부를 하던 석봉은 어머니가 너무 그리워 집을 찾아갔지요. 돌아온 석봉을 본 어머니는 말씀하셨습니다.

“나는 떡을 썰 테니, 너는 글을 쓰거라.”

불을 끈 어머니는 떡을 썰기 시작합니다. 어머니는 내일 아침에 아들에게 떡국을 끓여주려고 길다란 떡을 써셨던 것 같습니다. 그동안에 석봉은 지금까지 연마한 실력을 총동원하여 1부터 10 까지 숫자를 씁니다.

1, 2, 3, ...

이 상황을 파이썬으로 만들려면 어떻게 하는 것이 좋을까요? 지금까지 배웠던 것만 가지고 해볼까요?

```
>>> print(1)
1
>>> print(2)
2
>>> print(3)
3
>>>
```

좀 귀찮기는 해도 쓸만하지요? 그런데, 어머니께서 1부터 100 까지 써보라고 하십니다. 아, 막막합니다. 어느 세월에 다 씁니까. 100 까지 쓰면 아침이 밝아오겠군요.

이럴 땐 다른 방법을 찾아야겠지요? 오늘 그 비밀을 알려 드리려고 합니다.

바로 while 이라는 녀석입니다.

while 문

```
1, 2, 3, ..., 98, 99, 100
```

1부터 100까지는 저런 모양이 될 텐데요, 가만 보면 다음에 나오는 숫자는 앞의 숫자보다 1이 더 큽니다. 다시 말하면 앞의 숫자에 1을 더하면 다음 숫자가 나온다는 것이지요. 그러니까 계속 앞의 숫자에 1을 더 해나가다가 100까지 쓰고 그만두면 되는 겁니다.

다음을 잘 보세요.

```
>>> num = 1
>>> while num <= 100:
...     print(num)
...     num = num + 1
...
```

우리가 쓸 숫자를 num이라고 했고, 여기에 1을 넣어주었습니다. 이해가 잘 안 되시는 분은 변수 강좌를 다시 한번 보세요.

그다음에 while이라는 것이 나오죠? while은 우리말로'...하는 동안에'라는 뜻을 갖고 있죠? 여기서는'num이 100보다 작거나 같은 동안에'라는 뜻으로 쓴 것입니다.

아직 알쏭달쏭하시죠? 일단 다음 문장으로 넘어가보도록 하겠습니다.

```
print(num)      # 처음 네 칸을 띄어 써주세요
```

num이라는 변수에 들어있는 수를 화면에 뿌려달라는 것입니다. 지금까지의 강좌 내용을 이해하셨다면 이해하실 겁니다. 지금 num에는 1이 들어있으니 당근 1을 찍어주겠지요. while 블록 내부에 있다는 표시로 공백 네 칸을 써서 들여쓰기 해주세요.

그다음 문장을 봅시다.

변수값 증가

```
num = num + 1    # 위 줄과 들여쓰기를 맞춰주세요
```

이번엔 진짜로 이상한 것이 나왔습니다. num이 1이라면 $1 = 1 + 1$ 이 된다는 얼토당토 않은 소리군요. 그런데 프로그래밍에서 = 표시는'같다'는 뜻 말고 다른 뜻을 갖고 있습니다.

제 시계가 백만원이라는 것을 어떻게 표현했지요?

그렇죠, `watch = 1000000`입니다. 이것은 `watch`라는 변수에 1000000이라는 값을 넣어주라는 뜻이었습니다. 조금 전에 `num = num + 1`이라고 쓴 것은 `num`이라는 변수가 가진 값에 1을 더해서 다시 `num`에게 넣어주라는 의미입니다.

그래도 이해가 안 되신다면 다른 걸 보여드리도록 하지요.

```
>>> a = 3  
>>> b = a + 2
```

위의 문장을 보세요. `b`의 값이 얼마가 되었을까요? 이제 이해가 가시겠죠?

`num = num + 1` 대신 `num += 1`로 써도 똑같은 일을 한답니다.

while 문 수행 과정

자, 설명을 드리다보니 이야기가 잠시 딴 데로 쌌습니다. 보시기 편하도록 앞의 while 문 예제를 다시 한번 써볼게요.

```
>>> num = 1  
>>> while num <= 100:  
...     print(num)  
...     num = num + 1  
...
```

`while`은 어떤 조건이 만족되는 동안 그 아래에 쓴 문장들을 반복하는 기능을 갖고 있습니다. 여기서는 `num`이 100이 될 때까지 `print(num)`과 `num = num + 1`을 반복하는 것이지요. 제가 반복을 해보겠습니다.

처음엔 `num` 값이 1이니까 100 보다 작습니다. 그렇다면 그다음 문장을 수행해야겠지요?

`print(num)`이니까 화면에 1을 찍고 `num = num + 1`해서 `num`은 2가 됩니다.

그리고는 다시 위의 `while`로 돌아가지요.

그러면 `num` 값이 2이므로 `print(num)`이 2를 찍고 `num = num + 1`해서 `num`은 3이 됩니다.

그다음엔 `num` 값이 3이므로 `print(num)`이 3을 찍고 `num = num + 1`해서 `num`은 4가 됩니다.

그다음엔 `num` 값이 4이므로 `print(num)`이 4를 찍고 `num = num + 1`해서 `num`은 5가 됩니다.

헉헉헉...

그다음엔, ... 혁혁혁...

그렇게 하다 보면 언젠가는 `num` 값이 99 까지 올라갑니다. 이번에도 100 보다는 작으니까 또 99 찍고, `num`은 드디어 100 되지요. 이제 또다시 `while` 문으로 갑니다. `while` 다음에 뭐라고 써있죠?

```
num <= 100:
```

그렇습니다. `num`이 100 보다 작거나 같을 때 조건을 만족하는 겁니다. 그러면 하던 일을 계속해야겠죠? `print(num)`하면 화면에 100 을 찍고 `num = num + 1`해서 `num`에는 101 이 들어갑니다. 그다음에 `while`을 만나면 이번엔 `num`이 100 보다 크니까 그다음의 문장을 수행하지 않고 끝이 나고야 맙니다.

썰렁 ~.

우리는 머리를 약간 굴리고 프로그램 네 줄만 치면 1 부터 100 까지가 아니라 백만, 천만, 억까지도 숫자를 쓸 수 있는 것입니다. 놀랍지 않습니까, 여러분? 믿기지 않으신다고요? 그럼 한 번 따라해보실까요?

while 문 실습

먼저 파이썬 프로그램을 띄우고, 처음 두 문장을 쳐보세요. `while` 문 마지막에 콜론(:)이 꼭 들어가야 하니 빼먹지 마시고요.

```
>>> num = 1
>>> while num <= 100:
... 
```

둘째 줄까지 치면 다음 줄에 점 세 개가 자동으로 나타납니다. `while` 문은 여러 줄로 구성되기 때문에 다음 줄을 계속 입력하라는 뜻으로 나타나는 것이지요.

그럼 다음 줄을 입력하겠습니다. 점 세 개 뒤에 바로 쓰지 마시고 공백 키 네 번, 또는 Tab 키를 한 번 눌러서 간격을 띄운 다음에 명령을 입력하세요. 파이썬에서는 여러 줄로 이루어진 명령을 입력할 때 둘째 줄부터는 반드시 들여쓰기를 해줘야하니까요.

```
...     print(num)
...     num = num + 1
... 
```

그렇게 셋째, 넷째 줄까지 치고 ...이 나오면 Enter 를 한 번 더 눌러주세요. 그러면 더 이상 입력할 것이 없다는 것으로 알고 while 문이 끝나게 됩니다. 제대로 따라하셨다면 순식간에 1 부터 100 까지 화면에 나타날 겁니다.

그리하여 석봉은 100 까지 숫자를 쓰고 그리운 어머니 품에서 편안히 잠들 수 있게 되었다는 말씀.

2.1.1 연습 문제: 입력받은 숫자만큼 반복하기 (while)

문제

`input()`으로 사용자로부터 정수를 한 개 입력받아, 그 숫자를 숫자 크기만큼 반복해서 출력하는 파이썬 스크립트를 작성하세요. 이때 출력 앞에 공백을 한 칸 주어서, 입력과 출력이 구분되게 합니다. 단, `while` 문을 사용하세요.

예 1

입력:

```
3
```

출력:

```
3  
3  
3
```

예 2

입력:

```
5
```

출력:

```
5  
5
```

```
5  
5  
5
```

답

- 코드: [ch02/repeat_while.py](#)

2.1.2 연습 문제: 제곱표 (while)

문제

정수를 한 개 입력받아, 1부터 입력받은 수까지 각각에 대해 제곱을 구해 프린트하는 프로그램을 작성해 보세요. 단, while 문을 사용하세요.¹

예 1

입력:

```
3
```

출력:

```
1 1  
2 4  
3 9
```

예 2

입력:

```
5
```

출력:

```
1 1  
2 4  
3 9
```

¹C 프로그래밍: 현대적 접근에 소개된 내용으로 문제를 만들어 봤어요.

```
4 16  
5 25
```

답

- 코드: [ch02/square_table.py](#)

2.1.3 연습 문제: 양체공

문제

고무 공을 100 미터 높이에서 떨어뜨리는데, 이 공은 땅에 닿을 때마다 원래 높이의 $3/5$ 만큼 튀어오릅니다. 공이 열 번 훨 동안, 그때마다 공의 높이를 계산합니다.¹

- 1 60.0
- 2 36.0
- 3 21.599999999999998
- 4 12.95999999999999
- 5 7.775999999999999
- 6 4.665599999999995
- 7 2.799359999999996
- 8 1.679615999999998
- 9 1.007769599999998
- 10 0.604661759999998

round() 함수를 사용해서, 다음과 같이 소수점 아래 네 자리까지 출력해 보세요.

- 1 60.0
- 2 36.0
- 3 21.6
- 4 12.96
- 5 7.776
- 6 4.6656
- 7 2.7994
- 8 1.6796
- 9 1.0078
- 10 0.6047

¹제가 번역한 예 실린 문제가 재미있어서 가져온 것입니다.

round()는 다음과 같이 반올림을 해주는 함수입니다.

```
>>> round(1.23456, 2)      # 1.23456을 소수점 둘째 자리로(셋째 자리에서) 반올림  
1.23  
>>> round(1.23456, 3)      # 1.23456을 소수점 셋째 자리로(넷째 자리에서) 반올림  
1.235
```

참고로, **round(2.675, 2)**를 수행하면 결과가 2.68이 아닌 2.67로 나오는데, 이것은 버그가 아니라 부동소수점 (floating point) 연산의 한계입니다.

```
>>> round(2.675, 2)  
2.67
```

답

이 문제의 해답은 아래 주소에 있습니다.

- 코드: [ch02/ball.py](#)

답을 보지 말고 직접 풀어 보는 것이 좋겠죠! (파이썬 2를 사용하시는 분은 해답 코드에서 3/5를 3.0/5.0으로 바꿔서 해 보세요)

2.1.4 연습 문제: 코드를 보고 실행 결과 맞히기

문제

다음 코드¹를 읽고, 실행 결과를 알아맞혀 보세요.

```
number = 358

rem = rev = 0
while number >= 1:
    rem = number % 10
    rev = rev * 10 + rem
    number = number // 10

print(rev)
```

풀이

직접 실행해서 예상한 결과가 나왔는지 확인해 보세요.

- 코드: <http://bit.ly/3peNrR2>
- 풀이 영상: <https://youtu.be/OAxEFA44jU4>

¹edX.org 의 [Linux Basics: The Command Line Interface](#)에 나오는 연습 문제의 C 언어 코드를 파이썬으로 옮긴 것입니다.

2.2 조건문 (if-elif-else)

지금까지 저와 함께 파이썬을 알아가면서 어떤 생각이 드셨나요? 너무 쉽다는 분도 계실테고, 이런 것들 배워서 어디에 써먹는 건지 궁금한 분도 계실 것 같네요.

이 강의는 프로그래밍을 전혀 모르는 분을 위해 최대한 쉽게 쓰려고 했기 때문에 다른 언어를 접해보신 분에게는 지루할 것 같네요. 그런 분이라면 아마 여기까지 읽기 전에 다른 사이트를 찾아가셨겠죠?

또, 파이썬을 배워서 어디에 써먹느냐고 하신다면... 프로그램 만드는데 쓰지요. 웹사이트를 구축하는데도 씁니다. 지금 배우는 것과 같은 하찮은 것들이 모여서 엄청난 프로그램도 만들어 내는 것이지요. 조그만 레고 블록들이 모여서 큰 모형을 이루는 것과 같습니다. 차근차근 공부해가다 보면 점점 더 복잡한 프로그램을 만드실 수 있을 거예요.

그럼 또 새로운 것을 배워 볼까요? 이번엔 if 문입니다. If 는 '만약 ...이면' 이라는 뜻이지요? 파이썬에서도 같은 의미로 사용됩니다.

“달면 삼키고 쓰면 뱉는다.” 는 속담이 있지요. 그것을 파이썬에서는 쓰는 것과 비슷하게 써보겠습니다.

```
만 약  달 다 면 :  
    삼 킨 다 .  
그 렇 지  않 으 면 :  
    뱉 는 다 .
```

이번엔 영어를 조금 섞어서 써볼까요?

```
if  달 다 면 :  
    삼 킨 다 .  
else:  
    뱉 는 다 .
```

위에 든 예들은 설명을 위해서 써 본것인데, 그대로 작성하면 파이썬이 이해를 못합니다.

파이썬의 if 와 else

그럼 이번엔 실습을 해보겠습니다. 아래의 두 수 a 와 b 중에 어느 쪽이 더 클까요?

```
>>> a = 1234 * 4  
>>> b = 13456 / 2
```

if 문을 사용해서 a 가 크면 'a' 를 출력하고 b 가 크면 'b' 를 출력하도록 프로그램을 작성해 볼까요? 한번 따라서 쳐보세요.

```
>>> if a > b:  
...     print('a')  
... else:  
...     print('b')  
...
```

이후에 있는 것들은 주석 (설명) 이므로 입력하지 않아도 됩니다. a > b라고 쓴 것은 'a 가 b 보다 큰가?' 를 나타냅니다. 어렵지 않죠?

elif

조건을 여러 개 주는 것도 가능합니다. 이번엔 c 와 d 를 비교해 보겠습니다.

```
>>> c = 15 * 5  
>>> d = 15 + 15 + 15 + 15 + 15  
>>> if c > d:  
...     print('c is greater than d')  
... elif c == d:  
...     print('c is equal to d')  
... elif c < d:  
...     print('c is less than d')  
... else:  
...     print('I don\'t know')  
...  
c is equal to d
```

이렇게 **elif**라는 것을 사용하면 여러 개의 조건을 검사해서 그중에서 맘에 드는 것을 고를 수 있답니다.

== 연산자

여기서 새로운 것이 또 있는데, 바로 ==(등호 두 개) 입니다. ==(지금까지 알고 있던 =(등호 한 개) 와는 쓰임새가 다르니 혼동하지 않도록 주의하세요. `c == d`라고 쓰면 'c 와 d 의 값이 같은가?' 를 나타냅니다. 지금처럼 두 값을 비교할 때 사용하지요. 지금까지 등호 하나를 써서 `c = d`라고 쓴 것은 `d`의 값을 `c`에 넣으라는 뜻이었고요.

```
>>> watch = 1000000
```

기억나시죠? 이제 그 둘을 구별하실 수 있겠죠?

- if 문 입력 동영상 <https://youtu.be/pspPgQJ6CFE>

나머지 계산을 이용하는 if 문

어떤 수를 다른 수로 나눈 나머지가 0 이면 '나누어 떨어진다' 라고 합니다. 예를 들어, 48 을 4 로 나눈 나머지는 0 이므로, 48 은 4 로 나누어 떨어집니다.

```
>>> 48 % 4  
0
```

어떤 수 `a`가 다른 수 `b`로 나누어 떨어지는지를 파이썬의 if 문으로 다음과 같이 평가할 수 있습니다.

```
>>> a = 48  
>>> b = 4  
>>> if a % b == 0:  
...     print(f'{a}는 {b}로 나누어 떨어집니다.')  
... elif a % b != 0:  
...     print(f'{a}는 {b}로 나누어 떨어지지 않습니다.')  
...  
48는 4로 나누어 떨어집니다.
```

위의 예에서 `elif a % b != 0:` 대신 `else:`라고 해도 결과는 같겠죠?

조건에 따라 반복문 중단하기

어릴 때는 큰 수를 잘 이해하지 못하죠?

하나부터 열까지밖에 모르는 아이처럼, 10 보다 큰 숫자가 들어오면 멈추는 반복문을 작성해볼까요?

```
# filename: ten.py

max = 10

while True:
    num = int(input())
    if num > max:
        print(num, 'is too big!')
        break
```

이와 같이 반복문에서 **break**를 사용하면 빠져나올 수 있답니다.

입력:

```
3
6
9
12
```

출력:

```
12 is too big!
```

오늘의 강의는 여기까지입니다. 강의는 이해가 된다고 해서 그냥 훑어보지 마시고 꼭 예제를 따라서 쳐보시기 바랍니다. 그리고 그것과 비슷한 프로그램을 스스로 만들어 보세요. '백타가 불여일작'이라는 말도 있거든요.

백 번 따라해보는 것보다 한 번 직접 만들어 보는 것이 낫다

끝...

2.2.1 연습 문제: 숫자 읽기 (1~3)

문제

input()을 사용해 사용자로부터 입력받은 숫자를 한글로 출력하는 프로그램을 작성하세요. 단, 사용자는 1 이상 3 이하의 정수 중 하나를 입력한다고 가정합니다.

예 1

입력:

```
1
```

출력:

```
일
```

예 2

입력:

```
3
```

출력:

```
삼
```

답

- 코드: [ch02/korean_1_to_3.py](#)

2.2.2 연습 문제: 나이에 따른 세대 구분 (1)

문제

input()으로 사용자의 나이를 입력받은 후, 다음 표의 어느 세대에 속하는지 출력하세요. 입출력 문구는 자유롭게 지으면 됩니다.

| 출생 연도 | 세대 ¹ |
|-----------|-------------------------------------|
| ~1924 | 가장 위대한 세대 (The Greatest Generation) |
| 1925~1945 | 침묵 세대 (The Silent Generation) |
| 1946~1964 | 베이비붐 세대 (baby boomer) |
| 1965~1980 | X 세대 (Generation X) |
| 1981~1996 | 밀레니얼 (millennial) |
| 1997~ | Z 세대 (Generation Z) |

예

```
$ python3 generations1.py  
What year were you born? 1945  
You're the Silent Generation.
```

```
$ python3 generations1.py  
What year were you born? 1946  
You're a baby boomer.
```

¹ 〈American Generations Fast Facts〉 , CNN

```
$ python3 generations1.py  
What year were you born? 1964  
You're a baby boomer.
```

```
$ python3 generations1.py  
What year were you born? 1965  
You're a Gen X.
```

답

- 코드: ch02/generations1.py

2.2.3 연습 문제: 단위 기호

길이, 부피, 무게나 금액을 표기할 때 1000 을 ‘k’ 로 표기하곤 합니다. 예를 들어, 3000m 는 3km 로 표기하는 것이 일반적입니다.

다음은 입력받은 숫자가 1000 보다 크면 1 자리부터 100 자리까지의 숫자를 생략하고 ‘k’ 를 붙여주는 파일을 썬 코드입니다. 파일명은 `affix.py`로 하겠습니다.

```
1 num = int(input())
2 result = str(num)
3
4 if num >= 1000:
5     result = str(num // 1000) + 'k'
6 elif num >= 0:
7     pass
8
9 print(result)
10
```

코드 설명:

- 1 번째 줄: 입력받은 문자열을 정수 (int) 로 바꿨습니다. 나중에 숫자끼리 비교할 때 쓰려고 그렇게 한 것입니다.
- 2 번째 줄: if 문의 계산 결과를 담을 변수를 `result`라는 이름으로 만들었습니다.
- 4~5 번째 줄: 숫자가 1000 이상인지 검사해서 결괏값 (`result`) 을 만듭니다. 1~100 자리 숫자를 생략하려고 정수의 나눗셈 (//) 을 이용했습니다.
- 6~7 번째 줄: 숫자가 (1000 보다 작고) 0 이상이면 아무 일도 하지 않습니다 (`pass`).
- 9 번째 줄: `result`를 출력합니다.

실행

명령 프롬프트나 터미널에서 `python affix.py` Enter 를 입력하고, 1000 Enter 를 입력하면 1k 가 출력됩니다.

```
>python affix.py  
1000  
1k
```

문제

- 백만 이상의 숫자를 입력받았을 때 1~10 만자리 숫자를 생략하고 ‘M’을 붙여서 출력하게 코드를 수정해보세요.

```
>python affix.py  
10000000  
1M
```

- 성공했다면, 그 이상도 구현해보세요 ([단위기호, 수사의 바른 표기 방법](#) 참조).

```
>python affix.py  
10000000000  
1G  
...
```

풀이

- [ch02/affix.py](#)

2.2.4 연습 문제: 양수만 덧셈하기

문제

`input()`으로 사용자로부터 입력받은 정수를 계속 더해나가다가, 음수가 입력되면 중단하고 그 전까지 계산한 값을 출력하는 파이썬 스크립트를 작성하세요.

예

입력:

```
1  
2  
3  
-1
```

출력:

```
6
```

예 2

입력:

```
50  
60  
70  
-100
```

출력:

```
180
```

답

- 코드: [ch02/sum_positive.py](#)

2.2.5 연습 문제: 윤년 판별하기

문제

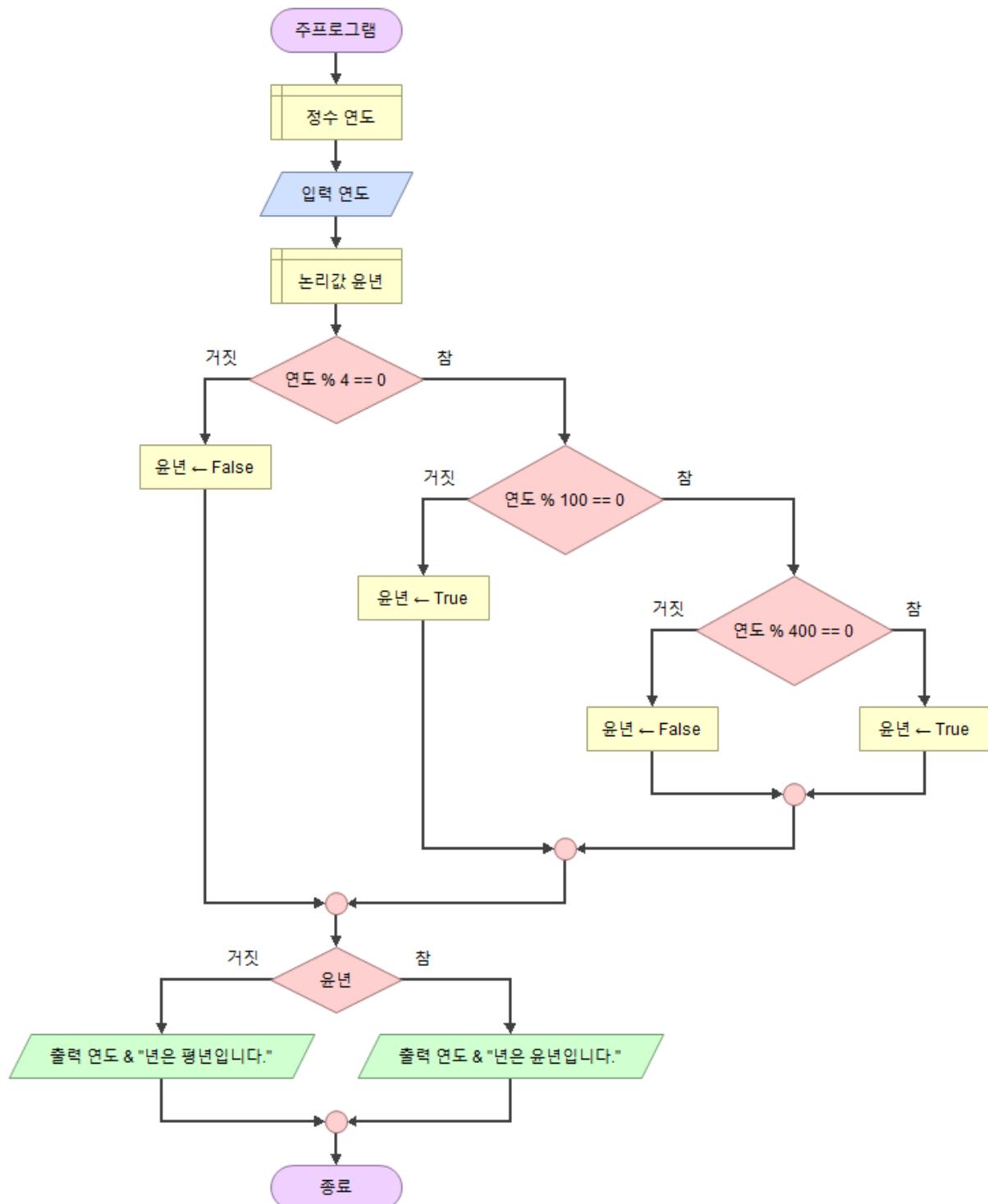
윤년은 역법을 실제 태양년에 맞추기 위해 여분의 하루 또는 월을 끼우는 해입니다. 현재 사용하는 그레고리력의 윤년 규칙은 다음과 같습니다.

1. 서력 기원 연수가 4로 나누어 떨어지는 해는 윤년으로 한다. (1988년, 1992년, 1996년, 2004년, 2008년, 2012년, 2016년, 2020년, 2024년, 2028년, 2032년, 2036년, 2040년, 2044년...)
2. 서력 기원 연수가 4, 100으로 나누어 떨어지는 해는 평년으로 한다. (1900년, 2100년, 2200년, 2300년, 2500년...)
3. 서력 기원 연수가 4, 100, 400으로 나누어 떨어지는 해는 윤년으로 둔다. (2000년, 2400년...)

출처: [위키백과](#)

이 규칙을 나타내는 흐름도를 그려보았습니다.¹

¹Flowgorithm을 사용해서 그렸습니다. Flowgorithm에 대해 더 알고 싶은 분은 [『흐름대로 프로그래밍하는 Flowgorithm』](#)을 참고하세요.



이 규칙에 따라, 연도를 나타내는 정수를 입력으로 받아서 윤년인지 아닌지 출력하는 프로그램을 작성해 보세요.

답

- 코드: [ch02/leap_year.py](#)
- 코드: 제가 풀이한 동영상을 유튜브에 올려 두었으니 참고하세요. <https://youtu.be/PZlyWHAvDCQ>

참고

- 제가 끈 것보다 좀 더 나은 풀이를 보내주신 독자가 있어서 [테스팅](#)에서 소개했습니다.
- 파이썬에는 주어진 해가 윤년인지 판별하는 함수가 있으므로 우리가 직접 구현하지 않아도 됩니다.
[모듈 사용법 알아내기](#) 연습 문제를 통해 알아봅니다.

2.2.6 and/or 연산자

이번에는 조건문에서 많이 쓰이는 **and**와 **or** 연산자를 알아볼게요.

if 문에 and/or 를 사용

if 문에 **and**, **or**를 사용할 수 있습니다.

예를 들어 보겠습니다. 다음과 같이 문자열 **s**가 있다고 할 때,

```
>>> s = 'banana'
```

다음과 같이 중첩된 if 문은,

```
>>> if 'a' in s:  
...     if 'b' in 'banana':  
...         print('banana에는 a도 있고 b도 있어요!')  
...  
banana에는 a도 있고 b도 있어요!
```

and를 써서 다음과 같이 바꿀 수 있습니다.

```
>>> if 'a' in s and 'b' in s:  
...     print('banana에는 a도 있고 b도 있어요!')  
...  
banana에는 a도 있고 b도 있어요!
```

중첩되었던 **if** 문을 **if** 한 번으로 줄였습니다.

여기서 퀴즈!

and 대신 **or**를 써서, ‘banana 에는 a 또는 c 가 있어요!’ 라고 출력하는 문장을 만들어 보세요.

if 문 없이 and/or 만 사용

사실, and 와 or 는 반드시 if 문에 들어가야만 하는 것은 아니고, 다음과 같이 and 또는 or 를 단독으로 사용해도 됩니다.

```
>>> 'a' in s  
True  
>>> 'b' in s  
True  
>>> 'c' in s  
False
```

이러한 True/False 값을 불 (**bool**) 이라고 하는데, 불값을 변수에 넣을 수도 있습니다.

```
>>> a_in_s = 'a' in s  
>>> a_in_s  
True
```

and/or 연산 순서

파이썬에서는 and/or 의 왼쪽 항을 먼저 평가 (계산) 하고, 오른쪽 항은 필요할 때만 평가합니다 (대부분의 고급 언어에서 이렇게 합니다).

예를 들어볼게요.

a 와 b 값이 다음과 같을 때,

```
>>> a = 3  
>>> b = 0
```

b 값이 0 이므로 b 를 분모로 하여 나눗셈을 하면 다음과 같이 **ZeroDivisionError** 가 발생합니다.

```
>>> a / b  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
ZeroDivisionError: division by zero
```

그런데 이 나눗셈 계산을 아래처럼 and 의 오른쪽 항에 넣으면 오류가 생기지 않습니다.

```
>>> (a * b) > 0 and (a / b) > 0  
False
```

파이썬이 ‘왼쪽 항을 평가해 보니, 오른쪽 항은 평가할 필요가 없겠구나’ 하고 넘어가 버린 것이죠.

위 문장의 순서를 바꿔서 나눗셈을 먼저 시켜보면 `ZeroDivisionError`가 발생하는 것을 볼 수 있습니다.

```
>>> (a / b) > 0 and (a * b) > 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

2.2.7 연습 문제: 나이에 따른 세대 구분 (2)

문제

미국과 달리, 우리나라에서는 보통 1955~1963년생을 ‘베이비붐 세대’로 봅니다.¹ 사용자가 한국인인지에 따라 세대를 구분할 수 있게 프로그램을 고쳐 보세요. (문제를 단순화하기 위해, 산업화 세대와 386 세대²는 고려하지 않습니다. 그리고 사용자는 미국인 또는 한국인이라고 가정합니다.)

예

```
$ python3 generations2.py  
What year were you born? 1954  
Are you Korean?(y/n) y  
You're the Silent Generation.
```

```
$ python3 generations2.py  
What year were you born? 1954  
Are you Korean?(y/n) n  
You're a baby boomer.
```

```
$ python3 generations2.py  
What year were you born? 1955  
You're a baby boomer.
```

```
$ python3 generations2.py  
What year were you born? 1963  
You're a baby boomer.
```

```
$ python3 generations2.py  
What year were you born? 1964
```

¹〈당사자도 헷갈리는 베이비붐 세대 기준...생물학에 사회·역사 혼합 때문〉, 브라보 마이 라이프

²〈산업화세대→베이비부머→X 세대→밀레니얼세대→Z 세대...세대별로 성장 배경과 소비 패턴·가치관이 모두 다르죠~〉, 생글생글

```
Are you Korean?(y/n) n  
You're a baby boomer.  
  
$ python3 generations2.py  
What year were you born? 1964  
Are you Korean?(y/n) y  
You're a Gen X.
```

문자열의 `lower()` 메서드는 영문 대문자를 소문자로 바꾼 값을 돌려줍니다.

```
>>> 'Yes'.lower()  
'yes'
```

문자열 뒤에 [0]을 붙이면 문자열의 첫 번째 글자를 얻을 수 있습니다.

```
>>> 'Yes'.lower()[0]  
'y'
```

풀이

저도 문제 만들고 풀면서 좀 헷갈렸는데요, 조건을 따지기 복잡할 땐 아래와 같이 표를 먼저 만들고 나서 조건 논리 (conditional logic) 를 세우면 좋습니다.

| 출생 연도 | 미국인 | 한국인(베이비 부머만 수정) |
|-----------|------------------------------------|------------------------------------|
| ~1924 | 가장 위대한 세대(The Greatest Generation) | 가장 위대한 세대(The Greatest Generation) |
| 1925~1945 | 침묵 세대(The Silent Generation) | 침묵 세대(The Silent Generation) |
| 1946~1954 | 베이비 부머(baby boomer) | 침묵 세대(The Silent Generation) |
| 1955~1963 | 베이비 부머(baby boomer) | 베이비 부머(baby boomer) |
| 1964 | 베이비 부머(baby boomer) | X세대(Generation X) |
| 1965~1980 | X세대(Generation X) | X세대(Generation X) |
| 1981~1996 | 밀레니얼(millennial) | 밀레니얼(millennial) |
| 1997~ | Z세대(Generation Z) | Z세대(Generation Z) |

저는 코드를 이렇게 작성했습니다.

- 코드: [ch02/generations2.py](#)

침묵 세대와 베이비붐 세대를 판정하는 코드에 국적을 묻는 부분을 추가했습니다.

베이비붐 세대 확인 코드를 설명드리면, 1963년생까지는 미국인과 한국인 모두 베이비붐 세대이므로

국적을 묻지 않아도 되고, 1964년생이면서 미국인일 경우 (한국인이 아닐 경우) 베이비붐 세대입니다.

```
elif y <= 1963 or ( # 
    y <= 1964 and # 
        input("Are you Korean?(y/n) ").lower()[0] != 'y' # 
):
    gen = 'a baby boomer' # 
```

참고로 파이썬에서 A or B 형태의 조건을 평가할 때는, A를 먼저 평가해서 A가 참이면 B를 평가하지 않습니다. 왜냐하면 A가 참이면, B가 참인지 아닌지에 관계없이 결과가 참이므로, B를 평가할 필요가 없기 때문입니다.

그리고 에서 != 대신 ==를 사용하려면 미국인이나고 물으면 됩니다.

사실 이렇게 헷갈리는 조건문을 만들면 버그가 생기기 쉽고 디버깅도 어려워서 추천하는 방식은 아니지만, 때로는 불가피하게 복잡한 조건을 따져야 할 때도 있으므로 그러한 연습을 할 수 있게 출제한 것입니다.

출생 연도를 먼저 묻고 국적은 필요할 때만 물어보게 하려다 보니 복잡해졌는데, 저처럼 하지 않고 국적을 먼저 묻고 나서 출생 연도를 물으면 조건문이 단순해져서 이해하기 쉬울 거예요.

2.3 for 를 사용하는 반복문

- 강의 영상: <https://youtu.be/TdFn4dnERHk>

이번엔 for 문에 대해서 알아볼 차례입니다. 파이썬에서 for 문의 쓰임새는 다른 언어와 차이가 있다고 하네요. 저도 그걸 모르고 한참 글을 쓰다보니 뭔가 이상하다는 것을 발견했습니다. ^^;

for 문은 우리가 전에 배웠던 리스트와 같은 **시퀀스** (sequence) 를 이용해서 원하는 명령을 반복할 때 쓰입니다. 시퀀스에 대해서는 나중에 자세하게 알려드리기로 하고, 전에 배웠던 리스트를 다시 한번 볼까요?

```
>>> family = ['mother', 'father', 'gentleman', 'sexy lady']
```

저희 가족이 이랬었는데 기억나시지요? 그냥 따라 치지 마시고 여러분의 가족을 나타내는 리스트를 만들어 보세요.

for 문

다음은 for 문을 이용해서 저희 가족들의 이름과 문자열 길이를 출력하는 프로그램입니다.

```
>>> for x in family:      # family의 각 항목 x에 대하여:  
...     print(x, len(x))    # x와 x의 길이를 출력하라.  
...
```

답은 아래와 같이 나오게 되지요.

```
mother 6  
father 6  
gentleman 9  
sexy lady 9
```

in family for x: 라고 쓰면 안되냐고요?

안되네요. -;

문법이 그런 거니까 그대로 써주시면 됩니다.

range()

이번엔 **range()**라는 것을 배워보도록 하지요. **range**는 범위라는 뜻인데 여기서는 어떤 정수를 인자로 주면 그 범위 안의 정수들을 만들어줍니다. 말은 좀 어렵지만 별 거 아니랍니다.

```
>>> list(range(2, 7))      # 파이썬 3  
>>> range(2, 7)          # 파이썬 2
```

이렇게 쳐 보세요. 어떤 답이 나오나요?

```
[2, 3, 4, 5, 6]
```

예, 2 이상 7 미만인 숫자로 리스트를 만들어 주었군요. 위에서 설명한 말이 이해되시죠?

그런데, **for**를 설명하다가 갑자기 웬 **range()**가 나오는 걸까요? 그렇습니다. **for** 문에 **range()**를 사용할 수 있습니다.

```
>>> a = [4, 5, 6, 7]  
>>> for i in a:  
...     print(i)  
...
```

위의 리스트를 사용한 예제와 아래의 **range()**를 사용한 예제는 출력이 같습니다.

```
>>> for i in range(4, 8):  
...     print(i)  
...
```

답이 어떻게 나올까요? 따라서 치시기 전에 먼저 생각을 해보세요. 그리 어렵지 않죠?

프로그래밍은 아무리 쉬운 것도 직접 해보지 않으면 자기 것으로 만들기 힘들답니다. 또, 생각 없이 책만 보고 따라한다고해서 빨리 늘지도 않습니다.

배우는 과정을 즐기면서 차근차근 연습하다보면 실력이 늘게 된답니다.

거북이

- Python 터틀 그래픽스와 반복문을 함께 사용하기 <https://youtu.be/WHRrp-t3a64>

2.3.1 연습 문제: 입력받은 숫자만큼 반복하기 (for)

문제

2.1.1 연습 문제와 같이, 사용자로부터 `input()`으로 정수를 한 개 입력받아, 그 숫자를 숫자 크기만큼 반복해서 출력하는 파이썬 스크립트를 작성하세요. 이때 출력 앞에 공백을 한 칸 주어서, 입력과 출력이 구분되게 합니다.

단, 이번에는 `for` 문을 사용하세요.

예 1

입력:

```
3
```

출력:

```
3
3
3
```

예 2

입력:

```
5
```

출력:

```
5
```

```
5  
5  
5  
5
```

답

- 코드: [ch02/repeat_for.py](#)

2.3.2 연습 문제: 제곱표 (for)

문제

연습 문제 2.2와 같이, 정수를 한 개 입력받아 1부터 입력받은 수까지 각각에 대해 제곱을 구해 프린트하는 프로그램을 작성해 보세요. 단, 이번에는 for 문을 사용하세요.

예 1

입력:

```
3
```

출력:

```
1 1  
2 4  
3 9
```

예 2

입력:

```
5
```

출력:

```
1 1  
2 4  
3 9  
4 16  
5 25
```

답

- 코드: [ch02/square_table_for.py](#)

2.3.3 연습 문제: 화학 실험실

문제¹

대학교의 화학자들은 상처를 매우 빠르게 치료하는 약물을 제조하는 새로운 과정을 개발했다. 제조 과정은 매우 길고 화학 약품을 매번 모니터링해야 하므로 몇 시간씩 걸린다! 학생들은 즐거나 딴짓을 하므로 이 일을 믿고 맏길 수가 없다. 그러므로 약물의 제조를 모니터링하는 자동 장치를 프로그래밍해야 한다. 장치는 15 초마다 온도를 측정해 프로그램에 제공한다.

프로그램은 먼저 최소와 최대의 안전 온도를 나타내는 두 개의 정수를 읽는다. 그 다음에, 장치가 제공하는 온도 (정수)를 계속 읽는다. 화학 반응이 완료되면 장치는 끝을 알리는 -999를 보낸다. 기록된 온도가 올바른 범위에 있을 경우 (최솟값 또는 최댓값과 같아도 된다) `Nothing to report`를 표시해야 한다. 하지만 온도가 위험 수준에 도달하면 `Alert!`를 표시하고 온도 측정을 중단한다 (장치가 온도값을 계속 보내더라도).

예 1

입력:

```
10 20
15 10 20 0 15 -999
```

출력:

```
Nothing to report
Nothing to report
Nothing to report
Alert!
```

¹edX 의 [C Programming: Language Foundations](#) 코스에 나온 문제입니다.

예 2

입력:

```
0 100  
15 50 75 -999
```

출력:

```
Nothing to report  
Nothing to report  
Nothing to report
```

`split()` 메서드를 사용해 문자열을 분할한 리스트를 얻을 수 있습니다.

```
>>> '0 100'.split()  
['0', '100']
```

다음과 같이 `split()`으로 얻은 결과를 바로 변수에 할당할 수 있습니다.

```
>>> freezing_point, boiling_point = '0 100'.split()  
>>> freezing_point  
'0'  
>>> boiling_point  
'100'
```

`input()`으로 문자열을 입력받을 때에도, 문자열을 분할했을 때 리스트 원소가 몇 개가 될지 미리 정해져 있다면 위와 같은 방법으로 변수에 할당할 수 있습니다. 원소 개수를 미리 알 수 없다면 `for` 문을 이용하세요.

답

저는 온도를 여러 번 측정한 값이 한 번에 입력되는 것이 아니라, 아래 그림처럼 측정할 때마다 따로 들어온다고 가정하고 풀었습니다.



The screenshot shows the Python 3.9.0 Shell window. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The shell area displays the following text:

```
>>>
= RESTART: C:/Users/sk8er/OneDrive/문서/GitHub/wikidocs-chobo-python/ch02/chemical_lab.py
10 20
15
Nothing to report
10
Nothing to report
20
Nothing to report
0
Alert!
>>>
= RESTART: C:/Users/sk8er/OneDrive/문서/GitHub/wikidocs-chobo-python/ch02/chemical_lab.py
0 100
15
Nothing to report
50
Nothing to report
75
Nothing to report
-999
>>>
```

The status bar at the bottom right indicates Ln: 1758 Col: 4.

- 코드: [ch02/chemical_lab.py](#)

아래 주소의 풀이도 참고하세요.

- <https://pybo.kr/pybo/question/detail/80/>

2.4 match-case 문

파이썬을 배우기 전에 다른 프로그래밍 언어를 배운 적이 있다면 switch-case 문법을 접하셨을 텐데요, 파이썬에는 그러한 구문이 없다가 파이썬 3.10 에 match-case 구문이 추가됐습니다. match-case 는 switch-case 와 비슷하기도 하지만 좀 더 강력한 기능이 있습니다.

홀수, 짝수 판별

우선 간단한 예부터 들어볼게요.

다음 코드는 1 부터 10 까지의 정수에 대해, 각각이 홀수인지 짝수인지를 출력합니다.

```
for n in range(1, 11):
    match n % 2:
        case 0:
            print(f"{n} is even.")
        case 1:
            print(f"{n} is odd.")
```

n을 2로 나눈 나머지가 0이면 짝수 (even number), 1이면 홀수 (odd number)라고 출력하는 것이죠.

결과:

```
$ python odd_even.py
1 is odd.
2 is even.
3 is odd.
4 is even.
5 is odd.
6 is even.
7 is odd.
8 is even.
9 is odd.
10 is even.
```

여기까지만 보면 switch-case 와 별반 다를 것이 없다고 생각하실 수 있는데요, 여기서 끝이 아닙니다. match-case 문을 잘 활용하면 상당히 간결하게 코딩할 수 있답니다.

피즈버즈

match-case 를 이용해 피즈버즈 (FizzBuzz) 문제를 풀어보겠습니다. 피즈버즈가 무엇인지 모르시는 분은 아래 링크한 글을 잠시 보고 오시기 바랍니다. if-else 문으로 직접 풀어보시면 더 좋습니다.

- <https://wikidocs.net/168132>

그럼, match-case 를 이용한 피즈버즈 풀이 코드를 소개합니다.

```
for n in range(1, 101):
    match (n % 3, n % 5):
        case (0, 0):
            print("FizzBuzz")
        case (0, _):
            print("Fizz")
        case (_, 0):
            print("Buzz")
        case _:
            print(n)
```

여기서 `match (n % 3, n % 5)`라는 구문을 보실 수 있는데, 소괄호로 둘러싸인 부분은 4 장에서 배울 튜플입니다. 튜플에 관해서는 나중에 배우기로 하고, 일단은 괄호 안에 `n % 3`과 `n % 5`라는 두 개의 식이 있고, 그 아래의 `case`에서 각각을 평가한다고 생각하시면 됩니다.

- `case (0, 0)`: 은 “`n`을 3 으로 나눈 나머지도 0 이고, 5 로 나눈 나머지도 0 인 경우” 를 가리킵니다. 따라서 ‘FizzBuzz’ 를 출력합니다.
- `case` 문에서 밑줄 (`_`) 은 아무 값이나 상관없다는 뜻입니다. 즉 `case (0, _)`: 은 “`n`을 3 으로 나눈 나머지가 0 인 경우” 입니다. 따라서 ‘Fizz’ 를 출력합니다.
- 마찬가지로 `case (_, 0)`: 는 `n`을 5 로 나누어 떨어지는 경우이고, ‘Buzz’ 를 출력합니다.
- `case _`: 는 그 밖의 모든 경우입니다. 3 으로도, 5 로도 나누어 떨어지지 않는 숫자들이 이에 해당합니다.

위 코드를 실행한 결과는 다음과 같습니다.

```
$ python fizzbuzz.py
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
```

Buzz
11
(이하 생략)

참고

- [『만들면서 배우는 러스트 프로그래밍』](#)에서 Rust의 match-case 문을 사용한 FizzBuzz 예제를 참고했습니다.

2.5 for-else 와 while-else

조건문에 **else**를 쓸 수 있다는 건 앞에서 살펴봤는데요, 파이썬에서는 반복문에도 **else**를 쓸 수 있답니다.

for-else

다음 for 문에서는 리스트의 원소를 차례대로 출력하고 나서, 모두 출력했다는 메시지를 출력합니다.

```
>>> for x in [1, 2, 3, 4]:
...     print(x)
... else:
...     print("리스트의 원소를 모두 출력했어요")
...
1
2
3
4
리스트의 원소를 모두 출력했어요
```

위의 코드만 보면, 마지막 출력을 굳이 **else** 블록에 넣지 않고 반복문 바깥에 두어도 될 것 같죠? 하지만 아래처럼 **break**가 등장하면 얘기가 달라집니다.

```
>>> for x in [1, 2, 3, 4]:
...     if x % 3:
...         print(x)    # x가 3의 배수가 아니면 출력
...     else:
...         break    # x가 3의 배수이면 반복문에서 빠져나감
... else:
...     print("리스트의 원소를 모두 출력했어요")
...
1
2
```

여기서는 반복문을 **break**했는데, 그럴 때는 **else** 블록이 실행되지 않습니다.

while-else

while 문도 마찬가지입니다. **while** 문이 **break**될 경우에는 **else** 블록이 실행되지 않습니다.

```
>>> countdown = 5
>>> while countdown > 0:
...     print(countdown)
...     countdown -= 1
...     if input() == '중단':
...         break
... else:
...     print('발사!')
...
5
4
3
중단
```

3. 함수

비슷비슷한 코드를 반복해서 작성하기 귀찮으신가요? 함수를 이용하면 반복적인 코드를 줄이면서, 코드를 사용하기 좋게 정리할 수 있어요.

이 장에서 배우는 것:

- [함수](#)
- [반환 문](#)
- 지역변수, 전역변수
- 람다



3.1 함수

지금까지는 코드를 한 줄, 한 줄 입력해서 결과를 보긴 했지만, 컴퓨터에게 일을 시키는 건지, 우리가 일을 하는 건지 헷갈릴 정도로 귀찮으셨을 거예요. 오늘 배우실 함수를 아시고 나면 프로그래밍이 좀 더 즐거워지지 않을까 싶네요. 그럼 시작해볼까요?

[1, 2, 3, 4, 5] 라는 리스트가 있다고 해볼게요. 이 리스트에는 원소가 몇 개 있을까요?

예, 5 개입니다.

이번엔 [3, 4, 62, 27, 83, 956, 26, 58, 3, 78, 168, 64, 78]이라는 리스트가 있다고 칩니다. 에구구... 원소가 넘 많으니까 a_list라는 이름을 붙여놓도록 하죠.

```
>>> a_list = [3, 4, 62, 27, 83, 956, 26, 58, 3, 78, 168, 64, 78]
```

이 리스트엔 원소가 몇 개 있을까요? 여기서 공부를 열심히 하셨던 분은 뭔가 생각이 나실 법도 한데...

len()이라는 것, 기억나세요?

아하, len()은 리스트에 들어있는 원소 개수, 그러니까 리스트의 크기를 알려주죠. 이 len()이 바로'함수'였던 것이랍니다.

아래의 예처럼 len() 함수를 쓰면 아무 리스트나 쉽게 크기를 알아볼 수 있죠.

```
>>> len([1, 2, 3, 4, 5])
5
>>> len(a_list)
13
```

만약 len() 함수가 없었다면, 우리는 리스트의 크기를 알고 싶을 때마다 복잡한 프로그램을 작성해야했을지도 몰라요. 함수가 있기 때문에 우리는 프로그래밍을 좀 더 쉽게 할 수가 있는 것이지요.

함수는 len()처럼 처음부터 파이썬에서 제공해주는 것도 있고, 우리가 필요로 하는 것을 직접 만들어 쓸 수도 있답니다. 또는 다른 누군가가 만든 함수를 얻어서 쓸 수도 있겠지요.

저는 처음에 함수라는 것을 배울 때 매우 어렵게 생각해서 애를 먹었는데, 사실은 이렇게 프로그래밍을 도와주는 고마운 존재라는 것을 나중에야 알게 되었답니다.

그렇다면 이번엔 함수를 직접 만들어 보면 어떨까요?

지난 강좌에서 리스트에 들어있는 원소를 차례대로 출력했던 것 기억나시죠? 그것과 같은 기능을 하는 함수를 한번 만들어보도록 하지요. 같이 따라해보세요.

```
>>> def print_list(a): # 지금부터 print_list 함수를 만들겠다는 뜻
...     for i in a:
...         print(i)
...
```

방금 우리는 `print_list()` 라는 함수를 만들었습니다. 간단하지요?

첫째 줄은 함수의 이름을 지어주는 부분이구요, 꽂호 안의 `a` 는 매개변수라고 합니다. 함수를 사용할 때는 `print_list([1, 2])` 와 같은 형태로 쓰면 된다는 것을 나타내지요. 이 때 `[1, 2]` 라는 리스트를 함수에 넣어주면 함수 내부에서는 `a=[1, 2]` 라고 생각하고 일을 하게 되구요.

둘째 줄부터는 어제 해본 것과 똑같죠. `a` 라는 리스트의 원소를 차례대로 출력하는 명령입니다.

우리가 만든 함수가 제대로 동작을 하는지 테스트를 해봅시다. 아까 만들어 둔 `a_list` 에 들어있는 원소들을 찍어볼까요?

```
>>> print_list(a_list)
```

함수를 제대로 만드셨다면 리스트에 들어있는 원소들이 주루룩 나열됩니다. 어떻습니까? 제법 쓸만하죠?

방금 만들어 본 함수에서는 매개변수를 받아서 일을 했었죠? 하지만 아래처럼 매개변수가 없는 함수도 만들 수 있답니다.

```
>>> def boy():
...     print('I am a boy.')
...     print('You are a girl.')
...
```

자, 이번엔 여러분이 직접 만들어볼 차례입니다. `a` 와 `b` 가운데 `a` 가 크면 '`a > b`' 라고 표시하고, `b` 가 크면 '`a < b`', 두 숫자가 같으면 '`a == b`' 라고 표시하는 함수를 만들어 보세요. 이 함수는 매개변수 두 개를 필요로 합니다. 함수를 만들 때 꽂호 안에 `(x, y)` 와 같은 형식으로 해주면 되겠죠? 답 다 가르쳐 드렸네요.^^

거북이

- 정사각형을 그리는 함수, 정삼각형을 그리는 함수: https://youtu.be/C6n_b7M7eFI
- 다각형을 그리는 함수: <https://youtu.be/h5bUrCQyom4>

3.1.1 연습 문제: 자릿수를 구하는 함수 만들기

`example.com`처럼 우리가 보통 ‘홈페이지 주소’라고 부르는 URL이 IP 주소로 변환된다는 것을 알고 계신가요?

주소를 192.0.2.10과 같이 나타내는 IPv4에서는 2564 개의 주소를 만들 수 있다고 해요.

```
>>> 256 ** 4  
4294967296
```

그러니까 여기 몇 개냐면

```
>>> len(str(_)) # 방금 구한 답(_)을 문자열(str)로 바꾼 것의 길이(len)  
10
```

10 자리 수이니까 4 뒤에 0이 9개 붙었다고 하면, 43억 개 정도 되네요.

2564은 232와 같습니다.

```
>>> 256 ** 4 == (2 ** 8) ** 4 == 2 ** 32  
True
```

요즘은 인터넷에 접속하는 기기가 너무 많다보니 주소가 얼마 남지 않아서, 아주 아주 아주 많은 주소를 만들 수 있는 IPv6로 바꿔 가고 있다고 합니다.

IPv6 주소는 2001:0db8:85a3:0000:0000:8a2e:0370:7334처럼 생겼고 2128 개의 주소를 만들 수 있습니다.

```
>>> 2 ** 128  
340282366920938463463374607431768211456  
>>> len(str(_))  
39
```

39 자리 숫자네요.

전 세계 인구를 80 억 명이라고 가정하면,

```
>>> 2 ** 128 / 8000_000_000  
4.253529586511731e+28
```

1 인당 4 양 (穰) 개씩 쓸 수 있다는 계산이 나옵니다.¹

문제

양 (陽)의 정수를 입력받아, 그 수가 몇 자리 숫자인지 출력하는 함수 `numOfDigits()`를 만들어 보세요.

예

```
>>> numOfDigits(12345)  
5  
>>> numOfDigits(1234567890)  
10
```

풀이

- 코드: `ch03/numdigits.py`

¹큰 숫자를 읽는 법은 https://ko.wikipedia.org/wiki/%EB%85%B8_%EB%A8%BC_%EB%A8%AC에 나오는 표의 ‘이만체진’ 열을 참고하세요.

3.1.2 연습 문제: 구구단

짜잔 ~! 반갑습니다. 여러분 ~. 제 강좌를 찾아주셔서 정말 기뻐요. 게시판에 올리시는 글을 보는 것이 낙 이랍니다. 전 요즘 학교 시험공부랑 과제물, 회사일로 조금 바쁘답니다 (마음만 바쁘지 TV 보고, 잠 잘 자고, 딴 짓도 많이 한답니다 ^^;).

오늘은 구구단을 준비했습니다. 구구단은 어느 언어든 프로그래밍 배울 때 빠지지 않는 약방의 감초입니다. 저는 초등학교 2 학년 때 그네 타면서 구구단을 외웠던 기억이 나는데, 지금은 몇 학년 때 배우는지 궁금하군요. 저는 웬지 8 단이 어렵더군요. 어머니는 뭐가 어렵냐고 하셨지만... 그래서 산수 시간에 곱셈문제를 풀 때는 가능하면 덧셈으로 바꿔서 풀곤했지요.

문제

다음 예와 같이 구구단을 2 단부터 9 단까지 계산해서 출력하는 프로그램을 짜보세요. 지금까지 배운 내용만 제대로 이해하시면 충분히 하실 수 있답니다.

예

출력:

```
2 * 1 = 2
2 * 2 = 4
...
9 * 9 = 81
```

힌트

음, 힘드십니까...

모르겠다고 하는 분이 정상입니다...

할 수 있다고 하는 분은 아마 예전에 비슷한 것을 배웠거나 머리가 좋은 분일 거예요.

그럼 정상인을 위해서 힌트를 드리겠습니다. 사실 힌트랄 건 없구요, 전체를 한 번에 만들기 힘들면 조금씩 나눠서 해보시라는 겁니다.

우선 2 단만 똑같이 만들어보세요. 이 강좌는 더 이상 보지 마시고 지금까지 배웠던 것을 참고해서 만들어보세요.

풀이

한번 해보셨나요? 해보신 분은 제가 짠 것과 비교해보세요.

2 곱하기 1을 출력

우선 2 곱하기 1부터 출력해보겠습니다. 단수를 변수 `m`으로, 곱하는 수를 `n`으로 하겠습니다.

```
>>> m = 2
>>> n = 1
>>> print(m, '* ', n, '=', m*n)
2 * 1 = 2
```

`print()` 함수에는 변수 `m`, 문자열 '`*`', 변수 `n`, 문자열 '`=`', 그 다음에 `m*n`이라는 식을 쉼표로 구분해 차례로 넣어줬습니다.

`print()` 함수의 인자가 너무 많아 타자 치기도 번거롭고 눈에 잘 들어오지 않으므로 다음과 같이 f-문자열을 사용하는 것이 낫겠네요. f-문자열은 파이썬 3.6 이상에서 사용할 수 있습니다.

```
>>> print(f'{m} * {n} = {m*n}')
2 * 1 = 2
```

2 단을 출력

다음은 구구단의 2 단을 출력하는 코드입니다.

```
>>> m = 2
>>> for n in range(1, 10):
```

```
...     print(f'{m} * {n} = {m*n:2d}')
...
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
```

변수 `m` 값은 아까처럼 숫자 2로 했지만, 변수 `n`은 `for` 문과 `range()` 함수를 이용해 값을 증가시켜면서 2 단을 출력하게 했습니다. `print` 문에서는 `m`과 `n`을 곱한 결과의 자리수를 맞추려고 f-문자열의 세 번째 중괄호에 `2d`를 지정했습니다.

한 단을 계산하는 함수

이번엔 한 단을 계산하는 함수를 만들어 보려고 합니다. 함수의 인자로 2를 넣으면 2 단을 출력하고, 5를 넣으면 5 단을 출력하는 거죠.

```
>>> def multi(m):
...     for n in range(1, 10):
...         print(f'{m} * {n} = {m*n:2d}')
```

아까 거랑 거의 비슷한데 코드를 전부 함수에 집어넣었죠? 그래도 훨씬 품 납니다. 한번 돌려보세요. 3 단 나와라 뚝딱!

```
>>> multi(3)
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
```

결과를 확인해 보셨나요? 쓸 만 하죠?

구구단 전체

이제 처음 문제도 풀 수 있을 것 같지 않나요? 위에서 만든 함수를 이용해 2 단부터 9 단까지 출력하면 되겠죠?

잘 되셨나요? 축하합니다.

사실 지금과 같은 경우에 굳이 함수를 만들 필요는 없었지요. 함수를 이용하지 않고 반복문 안에 또 반복문을 집어넣어도 된답니다. 함수에 대해 복습도 할 겸 이해하기 쉽도록 만든 것이지요. 반복문을 중첩하는 방법은 직접 해 보시면 좋겠습니다. 제가 코드를 보여드리지 않아도 잘 하시리라고 생각합니다.

그럼 모두 즐거운 하루 되시구요, 저는 다음 이 시간에...

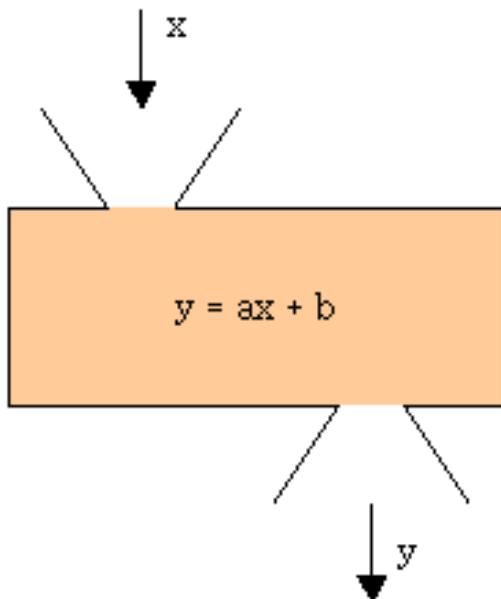
코드

- 코드: [ch03/multiplication_table.py](#)
- Flowgorithm 으로 작성한 구구단 흐름도: <https://wikidocs.net/167057>

3.2 반환 (return) 문

여러분, 함수가 무엇일까요? 지금까지 머리 아프게 함수를 공부했는데 또 무슨 소리냐고요?

물론 함수에 대해 계속 배워왔지만, 결정적으로 빠진 내용이 하나 있습니다. 사실은 우리가 초등학교 때부터 배워왔던 것이기도 합니다. 과연 그것이 무엇일까요?



이 그림 낯익으시죠? 함수에 x 를 집어 넣으면 함수가 주물럭주물럭 계산해서 y 라는 값을 돌려주는 그림입니다.

함수에 값을 넣으면 함수는 계산된 값을 돌려준다. 이것이 바로 함수의 핵심이지요. 프로그래밍에서도 마찬가지입니다. 지금까지 우리가 만든 함수들은 일은 열심히 하지만 돌려주는 것은 없었지요.

그렇다면 이젠 일도 하고 결과를 돌려주기도 하는 함수를 만들어 봐야겠죠?

```
>>> def f1(x):
...     a = 3
...     b = 5
...     y = a * x + b
```

```
...     return y          # y 값을 반환 한다
...
>>> c = f1(10)        # c = 35
>>> print(c)
35
```

위의 그림과 같은 역할을 하는 함수 `f1()`을 만들어봤습니다. 값을 돌려주기 위해 `return`이라는 것이 쓰였지요? 이렇게 만들어진 함수에 10이라는 인자를 넣어주면 함수는 35라는 값을 돌려줍니다. 따라서, 그 값을 다시 `c`라는 변수에 넣을 수도 있는 거죠.

만약, 함수를 정의할 때 `return y` 대신에 `print(y)`라고 썼다면 어떻게 될까요?

```
>>> def f2(x):
...     a = 3
...     b = 5
...     y = a * x + b
...     print(y)          # y 값을 출력 한다
...
>>> d = f2(10)        # d = ?
35
>>> print(d)
None
```

`d = f2(10)`이라고 하면 `f2()` 함수가 실행되어 35가 화면에 나타나지만 `d`에게 값을 반환하지는 않죠. 그래서 `d`를 프린트해보면 아무 값이 없다는 뜻으로 `None`이 출력됩니다.

어떠세요? 이제 함수가 값을 반환하는 것에 대해서 이해가 되시나요? 그렇다면 값을 반환하는 함수를 직접 만들어 보실 차례입니다.

삼각형의 넓이를 구하는 함수를 만들어보세요.

함수의 인자로는 삼각형의 밑변과 높이가 주어지고, 반환 (return) 값은 삼각형의 넓이가 되는 겁니다. 간단하겠죠?

문제를 풀어보신 분께는 신기한 것을 하나 알려드리겠습니다.

아직 안 풀어보셨으면 빨리 해보세요.

다 풀어보셨죠? 그럼, 알려드리도록 하겠습니다.

참과 거짓

1 더하기 1 은 2 맞죠? ‘참’, ‘거짓’으로 대답해보세요.

'참'이라고 대답하셨나요?

그럼 파이썬은 이 질문에 어떻게 대답할까요?

```
>>> 1 + 1 == 2  
True
```

참이라고 답을 하네요.

```
>>> 1 + 1 == 3  
False
```

이건 거짓이라고 하구요.

다음의 if 문을 보세요.

$1 + 1$ 이 2 가 맞으면 'yes' 라고 대답하고, 그렇지 않으면 'no' 라고 대답하겠죠?

```
>>> if 1 + 1 == 2:  
...     print('yes')  
... else:  
...     print('no')  
...  
yes
```

방금 알려드린 것과 함께 생각을 해보면

$1 + 1 == 2$ 라는 식이 True(참) 이면 yes를, False(거짓) 이면 no를 프린트한다는 걸 알 수 있습니다.

우리가 함수를 만들 때 이런 성질을 활용하면 도움이 되겠죠?

쉬운 덧셈 문제를 내는 함수를 만들어 보겠습니다.

```
>>> def quiz():  
...     ans = input('1 + 2 = ')  
...     return 1 + 2 == int(ans)  
...
```

`input()`이라는 함수는 사용자로부터 문자열 입력을 받는데 쓰이구요, `int()` 함수는 문자열을 정수로 바꿔줍니다. 예제에서는 `input()` 함수가 $1 + 2 =$ 이라는 문자열을 출력한 다음 사용자로부터 문자열을 입력 받아 그 값을 `ans`라는 변수에 넣어줬습니다.

셋째 줄에서는 $1 + 2$ 의 값과 `int(ans)`의 값이 같은지를 나타내는 `True`나 `False`로 반환하겠죠?

답을 맞히면 `True`를 돌려주고, 틀리면 `False`를 돌려주는 것이죠. 이해가 되시는지요?

위에서 만든 퀴즈를 다음과 같이 풀어볼 수 있습니다.

```
>>> quiz()
1 + 2 = 3
True
>>> quiz()
1 + 2 = 4
False
```

한번 테스트해보세요. 재미있죠?

우리가 배워온 것들이 점점 그럴 듯하게 모양을 갖춰가는 것 같네요.

오늘은 여기까지 ~ 수고하셨습니다 ~

3.2.1 연습 문제: 숫자 읽기 함수 (1~10)

문제

매개변수로 받은 정수를 한국어로 표기한 문자열을 반환하는 함수 `korean_number()`를 정의하세요.
단, 매개변수는 1 이상 10 이하의 정수라고 가정합니다.

예

```
>>> korean_number(1)
'일'
>>> korean_number(3)
'삼'
>>> korean_number(10)
'십'
```

답

- 코드: [ch03/korean_1_to_10.py](#)

3.2.2 연습 문제: 함수 정의하기

문제

문제 1

다음 `triple()` 함수를 완성하세요.

```
>>> def triple(x):
...     [REDACTED]
...
>>> triple(2)
6
>>> triple('x')
'xxx'
```

문제 2

오늘의 날짜 객체를 구하는 코드는 다음과 같습니다. (코드를 이해하지 못해도 이 문제를 풀 수 있습니다.)

```
>>> from datetime import datetime
>>> today = datetime.today()
>>> today
datetime.datetime(2021, 3, 21, 15, 46, 1, 94942)
```

위 코드의 `today`에서 연도를 구하는 방법은 다음과 같습니다.

```
>>> today.year
2021
```

태어난 해를 네 자리 숫자로 입력하면 한국 나이를 반환하는 함수 `korean_age()`를 작성하세요.

답

- 코드: [ch03/define_functions.py](#)

3.2.3 연습 문제: 이자 (단리) 계산

직장인 A 씨는 1년 동안 열심히 일해서 연말에 성과급으로 천만 원을 받았습니다. 연이율 3.875%(단리) 인 고정금리 상품에 예금하려고 합니다. 5년 동안 넣어두면 이자가 얼마 붙는지 계산해보겠습니다.

첫 해에 원금 10,000,000 원에 대한 이자 $10,000,000 * 0.03875 = 387500$ 원이 붙습니다.

둘째 해에 원금 10,000,000 원에 대한 이자 387500 원이 붙습니다.

마찬가지로 셋째, 넷째, 다섯째 해에도 해마다 같은 금액의 이자가 붙습니다.

만기가 되어 받을 수 있는 이자는 다음과 같습니다.

```
>>> 10000000 * 0.03875 * 5  
1937500.0
```

원금과 이자를 합한 총액, 즉 원리금은 다음과 같습니다.

```
>>> 10000000 + 10000000 * 0.03875 * 5  
11937500.0
```

소수점 이하는 필요 없지만 지금은 그대로 둘게요.

원금 (**P**rincipal), 이율 (**r**ate), 기간 (**t**ime) 이 주어졌을 때, 이자 (**I**nterest) 를 구하는 공식은 다음과 같습니다.

$$I = Prt$$

그리고 원리금 (**A**mount) 을 구하는 공식은 다음과 같습니다.

$$A = P(1 + rt)$$

문제

문제 1

원금 (`p`), 단리 이율 (`r`), 기간 (`t`) 이 주어졌을 때 **이자**를 구하는 함수 `simple_interest()`를 작성하세요.

예 1

```
>>> simple_interest(10000000, 0.03875, 5)
1937500.0
```

예 2

```
>>> simple_interest(1100000, 0.05, 5/12)
22916.666666666668
```

문제 2

원금 (`p`), 단리 이율 (`r`), 기간 (`t`) 이 주어졌을 때 **원리금**을 계산하는 함수 `simple_interest_amount()`를 작성하세요.

예 1

```
>>> simple_interest_amount(10000000, 0.03875, 5)
11937500.0
```

예 2

```
>>> simple_interest_amount(1100000, 0.05, 5/12)
1122916.666666665
```

답

- 코드: [ch03/simpleInterest.py](#)
- 구글 스프레드시트

참고

- Simple Interest Calculator $A = P(1 + rt)$, CaculatorSoup
- Business Math (Olivier), LibreTexts

3.2.4 연습 문제: 복리 계산

이번에는 복리 (複利, compound interest) 를 계산해 볼게요.

예를 들어 1,500,000 원을 3 개월 동안 넣어두면 연 4.3% 의 이자를 주는 상품이 있다고 할 때, 만기가 될 때마다 받는 이자와 원금을 합한 금액을 재예치하여 6 년간 운용했을 때 받는 총액을 계산해보겠습니다.

```
>>> r = 0.043  
>>> n = 4
```

위의 `r`은 연이율을 나타내는 변수고, `n`은 1년 동안에 복리가 몇 번 적용되는지는 나타내는 변수입니다. 3 개월이 4 번 지나야 1년이 되는데, 이때 4가 바로 `n` 변수값이 됩니다. 이렇게 하는 이유는 실제 계산을 해 보면 이해할 수 있습니다.

우선 첫 해만 계산해볼게요.

```
>>> 1500000 * (1 + r / n) # 처음 3개 월  
1516125.0  
>>> _ * (1 + r / n) # 그 다음 3개 월  
1532423.34375  
>>> _ * (1 + r / n) # 그 다음 3개 월  
1548896.8946953125  
>>> _ * (1 + r / n) # 그 다음 3개 월  
1565547.5363132872
```

이렇게 해서 처음 1년 동안의 원리금을 계산했습니다.

2년차도 계산해볼까요?

```
>>> _ * (1 + r / n)  
1582377.1723286551  
>>> _ * (1 + r / n)  
1599387.7269311883  
>>> _ * (1 + r / n)  
1616581.1449956987  
>>> _ * (1 + r / n)  
1633959.3923044025
```

같은 방식으로 6년치를 계산하면,

```
>>> _ * (1 + r / n)
1938836.8221341053
```

최종적으로 위 금액이 나옵니다. 한번 해보세요. 엄청 귀찮습니다. 소수점 이하가 길어서 보기 불편하지만 지금은 그대로 둘게요.

이 계산을 쉽게 하려면 다음과 같은 복리 계산 공식을 이용하면 됩니다.

$$P' = P\left(1 + \frac{r}{n}\right)^{nt}$$

P' : 원리금

P : 원금

r : 연이율

t : 기간

n : 복리 횟수

앞에서 손수 계산했던 것을 이번에는 공식에 대입해 풀어보겠습니다. 변수명은 모두 소문자로 할게요.

```
>>> p = 1500000
>>> r = 0.043
>>> t = 6
>>> n = 4
>>> p * (1 + r / n) ** (n * t)
1938836.8221341055
```

문제

복리 예금의 원금 (p), 연 이율 (r), 기간 (t), 복리 횟수 (n)에 대한 원리금을 계산하는 함수 `compound_interest_amount()`를 작성하세요.

예 1

6년간 매분기 이자를 주는 경우 ($t=6, n=4$)

```
>>> compound_interest_amount(1500000, 0.043, 6, 4)
```

```
1938836.8221341055
```

예 2

6년간 2년마다 이자를 주는 경우 ($t=6$, $n=1/2$)

```
>>> compound_interest_amount(1500000, 0.043, 6, 1/2)
1921236.0840000005
```

답

- 코드: [ch03/compoundInterest.py](#)
- 구글 스프레드시트

참고

- https://en.wikipedia.org/wiki/Compound_interest

3.3 지역변수, 전역변수

여러분 안녕하세요~.

오늘 (이장을 처음 쓴 날)은 제가 중간 시험 보는 날이랍니다. 시스템 프로그래밍이랑, 선형대수 과목이 구요, 시스템 분석 설계 과제물도 내야합니다.

시스템 프로그래밍은 컴퓨터의 CPU 구조랑, 어셈블리 언어, 어셈블리 언어를 해석해서 컴퓨터가 알 수 있게 기계어로 바꿔주는 어셈블러의 작동원리, 운영체제도 포함된 과목입니다. 한마디로 우리가 사용하는 프로그램과 컴퓨터 장치 사이에서 일하는 것이 시스템 프로그램이라고 할 수 있지요.

선형대수는 수학의 행렬, 벡터 같은 것에 대해 자세히 나오는데, 컴퓨터 그래픽을 구현할 때도 이런 것을 사용하더군요. 다른 여러 분야에도 응용이 되겠죠.

시스템 분석 설계는 건물을 짓기 전에 먼저 설계를 하는 것처럼 프로그램을 짤 때 전체적인 설계를 하는 것입니다.

보통 프로그래머가 되려면 자바, C, 비주얼 베이직 같은 프로그래밍 언어만 배우면 되는 것으로 생각을 하기 쉬운데, 사실 프로그래밍 언어는 한 부분에 불과하답니다. 그래서 좋은 프로그래머가 되기 위해선 많은 교육과 경험이 필요하지요.

제 강좌를 보고 계신 여러분 중에 프로그래머가 되고 싶은 분이 있다면 학교공부 (특히 영어, 수학) 도 열심히 하시고, 주위의 선배들, 관련 홈페이지나 월간지를 통해서 정보를 얻는다면 도움이 되실 거예요.

오늘도 함수에 관한 이야기입니다. 제목엔 변수라고 나와있지만요. 먼저 예제를 보실까요?

학교 다닐 때, 저희 학교 짱은 영구였습니다. 제가 전학가기 전까지는...

```
>>> jjang = '09'
```

제가 가서 바로 짱 먹었지요. 흐흐흐...

```
>>> jjang = 'pig dad'
```

애들한테 물어보면 누가 짱이라고 할까요? 당근...

```
>>> jjang  
'pig dad'
```

그런데 자기네 반에서 짱이라고 깑죽거리는 녀석이 있었으니, 바로 땅칠이라는 친구였답니다. 반이라는 함수를 만들어 보죠.

```
>>> def ban():  
...     jjang = '07'  
...     print('jjang =', jjang)  
...  
>>> ban()  
jjang = 07
```

그러나... 땅칠이도 제 앞에선 깨갱~ 이랍니다. 올 학교 짱은 변함없이 저걸랑요...

```
>>> jjang  
'pig dad'
```

ban() 함수 안에서 jjang = '07' 이라고 하면 jjang 이란 변수를 새로 만드는 거구요, 기존의 jjang 에는 영향을 미치지 않습니다. 그리고, ban 함수가 끝날 땐 그 함수 내에서 만들었던 변수들은 모두 없어지는 거죠.

이와 같이 함수 안에서 만들어진 변수를 지역변수라고 하고, 함수 밖에서 만들어진 변수를 전역변수라고 합니다. 지역변수는 함수가 호출되면 만들어져서, 함수의 실행이 끝날 때 함께 없어지는 반면, 전역변수는 함수와는 관계없이 항상 꿋꿋이 지구를 지킨답니다. 그래서 영어로 전역변수를 global 이라는 말로 표현하지용...

지역변수를 함수 밖에서 한번 불러볼까요?

```
>>> def d_is_10():  
...     d = 10          # 지역 변수  
...     print('d 값은 ', d, '입니다')  
...  
>>> d_is_10()  
d 값은 10 입니다  
>>> d  
Traceback (most recent call last):  
  File "<stdin>", line 1, in ?  
NameError: name 'd' is not defined
```

d 를 불러봐도 'd 라는 이름이 없다' 는 에러 메시지만 뜨지요? d_is_10() 함수가 실행되는 동안은 d 가 있

었는데, 함수의 실행이 끝난 다음에 함께 사라져버렸기 때문입니다. 반대로, 전역변수는 함수 안에서도 얼마든지 사용할 수 있답니다.

```
>>> x = 10                      # 전역 변수
>>> def printx():
...     print(x)
...
>>> printx()
10
```

그렇다면 지역변수 대신 전역변수만 쓰는 것이 편하겠다구요? 글쎄요... 전역변수는 프로그램이 복잡해 질수록 골치거리가 된답니다. 다른 엉뚱한 함수 때문에 변수의 값이 바뀌어버리는 수가 종종 있거든요. 그래서 필요에 따라 지역변수와 전역변수를 골라 쓰는 것이 좋답니다.

그리고, 함수 안에서 전역변수를 만드는 방법도 있답니다. 어떤 변수를 전역변수 (global)로 사용하겠다라고 명시해주는 것이죠.

```
>>> def e_is_10():
...     global e                      # 전역 변수
...     e = 10
...     print('e 값은 ', e, '입니다')
...
>>> e_is_10()
e 값은 10 입니다
>>> e
10
```

여기서는 `e_is_10()` 함수가 실행되면서 `e`라는 전역변수가 만들어지고, 이 변수는 함수의 실행이 끝난 다음에도 없어지지 않습니다.

저는 그만 시험보러 가야겠네요. 즐거운 주말 보내세요 ~.

3.5 람다 (lambda)

오늘은 람다 형식과 그것을 이용하는 여러 가지 함수들에 대해서 알아보겠습니다. 당장 완벽하게 소화하실 필요는 없을 것 같구요, 가벼운 마음으로 이런 것이 있다는 정도만 아셔도 되지 않을까 합니다. 람다 형식은 인공지능 분야나 AutoCAD 라는 설계 프로그램에서 쓰이는 Lisp 언어에서 물려받았다고 하는데요, 함수를 딱 한 줄만으로 만들게 해주는 훌륭한 녀석입니다. 사용할 때는 아래와 같이 써주면 되지요.

```
lambda 매개변수 : 표현식
```

다음은 두 수를 더하는 함수입니다.

```
>>> def hap(x, y):
...     return x + y
...
>>> hap(10, 20)
30
```

이것을 람다 형식으로는 어떻게 표현할까요?

```
>>> (lambda x,y: x + y)(10, 20)
30
```

너무나 간단하죠? 함수가 이름조차도 없습니다. '그냥 $10 + 20$ 이라고 하면 되지' 라고 말씀하시면 미워 잉~.

몇 가지 함수를 더 배워보면서 람다가 어떻게 이용되는지 알아보도록 하죠.

```
## map()
```

먼저 map 함수를 볼까요?

```
map(함수, 리스트)
```

이 함수는 함수와 리스트를 인자로 받습니다. 그렇죠? 그리고, 리스트로부터 원소를 하나씩 꺼내서 함수

를 적용시킨 다음, 그 결과를 새로운 리스트에 담아준답니다. 말이 좀 복잡하죠? 그럴 때 예제를 보는 게 최고죠.

```
>>> map(lambda x: x ** 2, range(5))          # 파일 션 2  
[0, 1, 4, 9, 16]  
>>> list(map(lambda x: x ** 2, range(5)))    # 파일 션 2 및 파일 션 3  
[0, 1, 4, 9, 16]
```

위의 `map` 함수가 매개변수로 받은 함수는 `lambda x: x ** 2` 구요, 리스트로는 `range(5)` 를 받았습니다. `range` 함수는 알고계시죠? `range(5)` 라고 써주면 `[0, 1, 2, 3, 4]` 라는 리스트를 돌려줍니다. 그리고 `x ** 2` 라는 것은 `x` 값을 제곱하라는 연산자죠.

`map` 함수는 리스트에서 원소를 하나씩 꺼내서 함수를 적용시킨 결과를 새로운 리스트에 담아주니까, 위의 예제는 0 을 제곱하고, 1 을 제곱하고, 2, 3, 4 를 제곱한 것을 새로운 리스트에 넣어주는 것입니다.

위의 예제를 람다가 아닌 보통의 함수로 구현하면 어떻게 될까요?

reduce()

이번엔 `reduce` 함수를 살펴봅시다.

```
reduce(함수, 시퀀스)
```

형식은 위와 같구요, 시퀀스 (문자열, 리스트, 튜플) 의 원소들을 누적적으로 (?) 함수에 적용시킨답니다. 말이 진짜 어렵군요. 예제를 살펴보도록 하겠습니다.

```
>>> from functools import reduce      # 파일 션 3에서는 써 주셔야 해요  
>>> reduce(lambda x, y: x + y, [0, 1, 2, 3, 4])  
10
```

위의 예제는 먼저 0 과 1 을 더하고, 그 결과에 2 를 더하고, 거기다가 3 을 더하고, 또 4 를 더한 값을 돌려줍니다. 한마디로 전부 다 더하라는 겁니다. 생각보다 쉽죠?

하... 하... 하...

의기양양하게 '예'라고 대답하신 분들을 위해 짜증나는 예제를 권해드리죠~.

```
>>> reduce(lambda x, y: y + x, 'abcde')  
'edcba'
```

전 원래 뭐 하나를 배우면 꼭 엽기적인 실험을 해본답니다. 더 짜증나라고 설명 안 해드릴립니다~

filter()

그 다음은 filter 를 살펴볼 차례입니다. 필터가 뭐죠? 정수기에서 물을 걸러주는 것이 필터죠? 에어컨의 바람 들어가는 곳에도 필터가 달려있구요.

filter(함수, 리스트)

파이썬의 필터는 이렇게 생겼는데요, 리스트에 들어있는 원소들을 함수에 적용시켜서 결과가 참인 값들로 새로운 리스트를 만들어줍니다. 다음은 0 부터 9 까지의 리스트 중에서 5 보다 작은 것만 돌려주는 예제입니다.

```
>>> filter(lambda x: x < 5, range(10))      # 파이썬 2  
[0, 1, 2, 3, 4]  
>>> list(filter(lambda x: x < 5, range(10))) # 파이썬 2 및 파이썬 3  
[0, 1, 2, 3, 4]
```

lambda x: x<5 라고 쓰니까 왠지 수학책에서 본 듯한 느낌이 들지 않습니까? 수학자들이 파이썬을 좋아한다던데...

위의 예제가 어떻게 돌아가는지는 척 보면 아시겠죠?

0 부터 9 까지의 리스트에서 숫자를 하나씩 꺼냅니다.

그 숫자를 x 라 하고, x < 5 가 '참' 이면 살려줍니다.

살아남은 것들은 새로운 리스트에 넣어줍니다. 끝.

자, 이번엔 홀수만 돌려주는 filter 를 만들어 보도록 합시다.

먼저 홀수가 뭔지 생각해볼까요?

짝수는 2 로 나누어 떨어지는 수이고, 홀수는 2 로 나누어 떨어지지 않는 수입니다.

짝수를 2 로 나눈 나머지는 0 이고, 홀수를 2 로 나누면 나머지가 1 이죠.

또, 나머지를 구할 땐 %라는 연산자를 쓰면 됩니다. 예를 들어서, 50 을 8 로 나누면 몫은 6이고 나머지는 2니까 50 % 8 은 2 가 되는 거지요.

이제 홀수를 돌려주는 필터를 만들어 보겠습니다.

```
>>> filter(lambda x: x % 2, range(10))      # 파이썬 2  
[1, 3, 5, 7, 9]  
>>> list(filter(lambda x: x % 2, range(10)))  # 파이썬 2 및 파이썬 3  
[1, 3, 5, 7, 9]
```

지난 시간에 '참'은 1이고 '거짓'은 0이라고 했죠? 위의 filter 함수를 실행시키면,
0을 2로 나눈 나머지는 0이니까 람다 함수의 결과값은 0이고, 0은 '거짓'이니까 버려집니다.
1을 2로 나눈 나머지는 1이니까 람다 함수의 결과값은 1이고, 1은 '참'이니까 통과하지요.

...

이런 식으로 수행하면서 홀수만 둘려주게 되는 거지요.

휴 ~ 이번 강좌는 설명하기 힘들어서 쓰면서 한참 애먹었네요. 잘 이해되지 않는 부분이 있으면 댓글 달아주세요. 그럼 안녕 ~

3.5.1 연습 문제: 놀이 공원 (1)

둘리와 도우너, 마이콜이 놀이 공원에 갔습니다. 놀이 기구 중에는 탑승자의 키를 제한하는 것이 있네요.

문제

놀이 기구의 이름과 키 제한을 나타낸 문자열을 입력받아서, 놀이 기구의 이름, 탑승 가능한 키의 하한(下限)과 상한(上限)을 각 행에 출력합니다.

코드는 다음과 같이 작성하며, ch03 폴더 아래에 파일명을 `ridereader.py`로 저장합니다.

```
def read(text):
    # 이 곳에 코드를 작성하세요.
    return ridename, cmmin, cmmax

if __name__ == "__main__":
    ridename, cmmin, cmmax = read(input())
    print("이름:", ridename)
    print("하한:", cmmin)
    print("상한:", cmmax)
```

입출력

예 1

입력:

```
와일드 윙: 110cm 이상
```

출력:

```
이름: 와일드 윙  
하한: 110  
상한: None
```

예 2

입력:

```
톰 오브 호러: -
```

출력:

```
이름: 톰 오브 호러  
하한: None  
상한: None
```

예 3

입력:

```
플라이벤처: 140cm~195cm
```

출력:

```
이름: 플라이벤처  
하한: 140  
상한: 195
```

문자열의 `split` 메서드를 써서, 구분자 (delimiter)를 기준으로 문자열을 분할할 수 있습니다. 다음 예에서는 콜론 (:)을 구분자로 삼았습니다.

```
>>> hms = "13:48:03"  
>>> hms.split(':')  
['13', '48', '03']
```

`str.strip`으로 문자열 앞뒤의 공백을 제거할 수 있습니다.

```
>>> str.strip(" I am a boy. ")  
'I am a boy.'
```

답

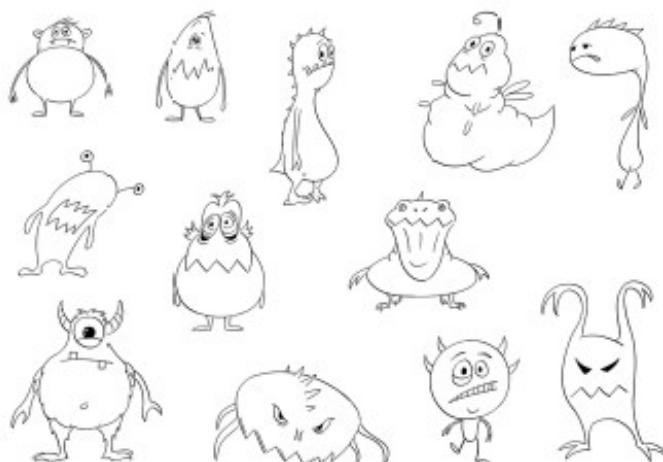
- [ch03/ridereader_nolambda.py](#) (for 문을 사용)
- [ch03/ridereader.py](#) (람다를 사용)

4. 데이터 타입

파이썬의 여러 가지 데이터 타입 (data type) 을 알아봅니다.

이 장에서 배우는 것:

- 자료형
- 문자열 (str) 과 리스트 (list)
- 튜플 (tuple)
- 딕셔너리 (dict)
- 세트 (set)



4.1 자료형

여러분 이진수에 대해서 알고 계시지요? 아마 중학교 때 배웠던 것 같네요. 아직 배우지 않은 분은 그런 것 이 있다는 것만 알고 계시구요. 컴퓨터에서는 이진수가 중요합니다. 왜냐하면 컴퓨터 내부에서는 모든 정보를 이진수로 처리하기 때문이죠. 예를 들어서 65 라는 숫자를 컴퓨터 내부에서는 이진수 01000001 로 처리합니다. 우리가 $65 + 30$ 이라고 명령을 내리면 컴퓨터는 그것들을 모두 이진수로 바꿔서 계산을 한다 음에 그 결과를 다시 우리가 쓰는 십진수로 바꿔주는 거죠. 컴퓨터는 숫자 뿐만 아니라 문자라든지, 제 아 무리 복잡한 정보도 모두 2 진수로 처리한답니다.

컴퓨터에서 영어를 사용하기 위해서는 알파벳 한 자 한 자마다 숫자로 번호를 매겨서 처리를 하지요. 알 파벳에 번호를 붙이는 규칙 중에서 널리 쓰이는 것으로 ASCII(아스키) 라는 규약이 있습니다. 아스키에서는 알파벳 A 를 숫자 65 로 표현하는데요, 어차피 숫자 65 는 다시 이진수로 바꿔서 처리하겠죠?

그렇다면 여기서 이상한 점이 생깁니다. 이진수 01000001 이 있는데 컴퓨터는 이것이 숫자 65 인지, 아니면 문자 A 인지 어떻게 알 수 있을까요?

너무 어렵게 생각하실 건 없답니다.

숫자인지 문자인지 표시를 해주면 되는 거죠. 사람이 해주든지, 컴퓨터가 알아서 하든지 말입니다.

그렇게 표시를 해주는 것이 바로 자료형이라고 할 수 있습니다.

지금까지 우리가 자료형에 대해서 잘 몰라도 프로그램을 짤 수 있었던 것은 우리가 자료를 만들 때마다 파이썬에서 자동으로 자료형을 정해주었기 때문인데요, 다른 프로그래밍 언어에서는 프로그래머가 직접 정해주어야 하는 경우도 있습니다. 그리고 프로그래밍 언어마다 제공해주는 자료형에 차이가 있지요.

type() 함수를 사용하면 자료형을 쉽게 확인할 수 있습니다.

```
>>> type(6)                                # 정수  
<type 'int'>  
>>> type('A')                             # 문자열  
<type 'str'>
```

그렇다면 파이썬에는 어떤 자료형이 있는지 살펴볼까요?

파이썬의 자료형은 크게 숫자 (numbers), 시퀀스 (sequence), 맵핑 (mapping) 등으로 나눌 수 있습니다.

숫자

숫자를 나타내는 자료형으로는 정수 (**int**), 부동소수점수 (**float**), 복소수 (**complex**) 가 있습니다.

int

int는 정수 (integer) 를 나타냅니다.

```
>>> type(1000000000)          # 정수  
<class 'int'>
```

정수가 너무 길어서 읽기 힘들면 밑줄을 넣어도 돼요.(파이썬 3.6 이상)

```
>>> 100_000_000    # 세 자리마다  
1000000000  
>>> 1_0000_0000    # 네 자리마다  
1000000000
```

float

float는 원래 부동소수점수 (floating-point number) 를 가리키는데, 지금은 단순히 소수점 이하를 표현할 수 있는 수라고 생각하셔도 좋습니다.

```
>>> type(2.8)                # 부동 소수 점수  
<type 'float'>
```

1장에서 봤듯이, int 끼리 연산한 결과가 float 로 나오기도 해요.

```
>>> 5 / 3  
1.6666666666666667
```

complex

그리고 고등학교에서 배우는 복소수를 **complex**로 나타냅니다.

```
>>> type(3+4j)          # 복소수  
<type 'complex'>
```

프로그래밍 언어에 복소수라는 자료형이 있는 것은 파이썬에서 처음 봤네요. 제곱하면 -1 이 되는 수 i 를 ‘허수 (imaginary number)’라고 하죠.

$$i^2 = -1$$

파이썬에서는 허수 i 를 **j**로 나타냅니다.

```
>>> (1j) ** 2  
(-1+0j)
```

고등학생 이상 언니들을 위해 한 가지 예를 보여드리면, 복소수의 거듭제곱 $(1 + i)^{10}$ 을 손으로 계산하는 과정은 이렇게 됩니다.¹

$$(1 + i)^2 = 1 + 2i + i^2 = 2i$$

$$(1 + i)^{10} = \{(1 + i)^2\}^5 = (2i)^5 = 2^5 \cdot i^5 = 32(i^2)^2i = 32i$$

파이썬에서는 이렇게 할 수 있어요.

```
>>> (1 + 1j) ** 10  
32j
```

시퀀스

문자열 (**str**), 리스트 (**list**), 튜플 (**tuple**), 사용자 정의 클래스가 시퀀스에 속합니다.

```
>>> type("Love your Enemies, for they tell you your Faults.")  
<class 'str'>  
>>> type(['love', 'enemy', 'fault'])  
<class 'list'>  
>>> type(('love', 'enemy', 'fault'))  
<class 'tuple'>
```

for 문에서 사용할 수 있는 것들이 바로 시퀀스입니다.

¹김동식, 『알기 쉽게 풀어 쓴 기초공학수학』, 2022, 생능출판

문자열이 시퀀스에 속하네요. 여러 개의 문자를 한 줄로 세워뒀으니 그럴 법도 하겠죠?

튜플과 사용자 정의 클래스에 대해서는 뒤에서 설명드리지요.

문자열 슬라이싱

아래와 같이 문자열 인덱스를 이용해 문자열의 일부를 복사할 수 있습니다.

```
>>> p = 'Python'  
>>> p[0:2]  
'Py'
```

시작 인덱스가 0일 때는 아래처럼 콤마 앞의 0을 생략할 수도 있습니다.

```
>>> p[:2]  
'Py'
```

음수 인덱스를 사용해 문자열의 뒷부분을 복사할 수도 있습니다.

```
>>> p[-2:]  
'on'
```

다음과 같이 콤마의 앞뒤 숫자를 모두 생략하면 문자열 전부를 복사할 수 있습니다.

```
>>> p[:]  
'Python'
```

역순으로 복사하는 것도 가능합니다.

```
>>> p[::-1]  
'nohtyP'
```

매핑

딕셔너리 (**dict**) 는 키 (key) 와 값 (value) 의 짹으로 이루어집니다. 이런 것을 매핑이라고 합니다.

```
>>> type({'one': 1, 'two': 2, 'three': 3})  
<class 'dict'>
```

불

참, 거짓을 표현하는 불 (**bool**) 도 있습니다.

```
>>> type(False)
<class 'bool'>
>>> type(3 >= 1)
<class 'bool'>
>>> type(True == 'True')
<class 'bool'>
```

세트

집합을 표현하는 세트 (**set**) 도 있습니다.

```
>>> fruits = {'apple', 'banana', 'orange'}
```

세트는 원소의 순서가 유지되지 않고 중복 원소를 갖지 않는 ‘집합’으로서의 특징이 있으며, 집합 연산을 사용할 수 있습니다.

강좌가 횟수를 거듭할수록 머리가 복잡해지네요. 제가 혼자 공부했으면 흐지부지할 때가 된 거지요. 그러나 여러분이 지켜보고 계신 한 포기는 없습니다. 제가 바로 이런 점을 노리고 혼자 공부하지 않고 굳이 강좌를 올리는 것입니다. 흐흐흐...

그러니 여러분도 힘내세요. 우리 함께 파이썬 정복하는 그 날까지 ~

4.1.1 연습 문제: 회문 판별 함수 만들기

문제

거꾸로 배열해도 같은 단어 혹은 문장이 되는 것을 **회문** (palindrome) 이라고 합니다. 다음은 회문의 예입니다.

- Anna
- Civic
- Kayak
- Level
- ...

문제 1

주어진 단어가 회문인지 판별하는 함수 `palindrome()`을 작성하세요. 단, 문자열 입력은 모두 소문자로 이뤄지며 공백을 포함하지 않는다고 가정합니다.

```
>>> palindrome('anna')
True
>>> palindrome('banana')
False
```

문제 2

대문자와 소문자가 섞여 있더라도 회문으로 판정하도록 함수를 개선하세요.

```
>>> palindrome('Anna')
True
```

문제 3

공백이 섞여 있더라도 회문으로 판정하도록 함수를 개선하세요.

```
>>> palindrome('My gym')
True
```

문자열을 [슬라이싱](#) 해서, 문자열에 속한 문자들을 거꾸로 배열한 문자열을 얻을 수 있습니다.

```
>>> 'Python'[::-1]
'nohtyP'
```

문자열의 [lower\(\)](#) 메서드를 이용해 소문자만으로 이루어진 문자열을 얻을 수 있습니다.

```
>>> 'Python'.lower()
'python'
```

그리고 [replace\(\)](#) 메서드를 이용해 문자열 일부를 다른 문자열로 바꾼 문자열을 얻을 수 있습니다.

```
>>> 'Python'.replace('P', 'J')
'Jython'
```

답

- 《흐름대로 프로그래밍하는 Flowgorithm》에서 회문 판별 함수의 흐름도를 볼 수 있습니다.
- 다음은 파이썬으로 간단하게 작성한 코드입니다. 위의 흐름도와 어떤 차이가 있는지 비교해보고, 코드를 개선할 수 있는지 생각해봅시다. [ch04/palindrome.py](#)

4.2 문자열과 리스트

문자열과 리스트를 좀 더 자세히 알아봅시다.

문자열

문자열에서는 요런 식으로 한 글자마다 번호를 매깁니다. 문자열을 만들어서 이것저것 시켜보세요.

```
>>> x = 'banana'  
>>> x[0]          # 0번 글자는?  
'b'  
>>> x[2:4]        # 2번부터 4번 앞(3번)까지는?  
'na'  
>>> x[:3]          # 처음부터 3번 앞(2번)까지는?  
'ban'  
>>> x[3:]          # 3번부터 끝까지는?  
'ana'
```

그렇다면 `banana`를 `nanana`로 바꿀 수는 있을까요?

```
>>> x[0] = 'n'
```

요렇게 해보면...

된다구요? 안 됩니다. 문자열에 들어있는 글자는 바꿀 수가 없답니다.

그래도 꼭 바꾸고 싶다면 이렇게 할 수는 있죠.

```
>>> x = 'n' + x[1:]  
>>> x  
'nanana'
```

이 방법은 `b`를 `n`으로 바꾼 것이 아니고 `n`과 `anana`를 합쳐서 `x`에 새로 넣어준 것입니다.

문자열에 어떤 글자가 몇 번째 자리에 있는지 알고 싶을 때는 `find()`를 사용하면 됩니다.

```
>>> s = "hello Python!"  
>>> s.find('P')  
6
```

'P' 가 6 번 인덱스에 있다는 것을 알았으니, 다음과 같이 [슬라이싱](#) 해서 다른 변수로 저장할 수도 있겠죠?

```
>>> s[0:6]  
'hello'  
>>> h = s[0:6]  
>>> h  
'hello'
```

위의 `h` 변수의 끝에는 공백이 포함되었는데, 다음과 같이 슬라이싱을 하거나 `rstrip()`으로 제거할 수 있습니다.

```
>>> h[0:5]  
'hello'  
>>> h.rstrip()  
'hello'
```

또는 다음과 같이 주어진 문자열을 분할한 리스트를 생성하는 `split()`을 이용해 첫 번째 단어를 알아내는 방법도 있습니다.

```
>>> s.split()  
['hello', 'Python!']  
>>> s.split()[0]  
'hello'
```

리스트

이번엔 리스트를 살펴보도록 하겠습니다. 원소를 추가하는 것부터 해볼까요?

```
>>> prime = [3, 7, 11] # 3, 7, 11을 원소로 갖는 리스트 prime을 만듦  
>>> prime.append(5) # prime에 원소 5를 추가  
>>> prime  
[3, 7, 11, 5]
```

`sort` 함수를 사용하면 정렬을 간단하게 할 수 있구요.

```
>>> prime.sort() # prime을 원소 크기 순으로 정렬
```

```
>>> prime  
[3, 5, 7, 11]
```

아차, 2를 빼뜨렸네요. 맨 앞(0번)에 2를 삽입(insert) 하겠습니다.

```
>>> prime.insert(0, 2)  
>>> prime  
[2, 3, 5, 7, 11]
```

원소를 삭제하는 것도 되지요. 4번 원소를 삭제해보겠습니다.

```
>>> del prime[4]           # prime의 4번 원소를 삭제  
>>> prime  
[2, 3, 5, 7]
```

원소를 삭제할 때 pop()을 사용할 수도 있습니다. pop()은 리스트에서 삭제한 원소를 반환(return) 하므로 변수로 받아서 나중에 사용할 수도 있죠.

```
>>> a = prime.pop() # 삭제한 원소를 a 변수로 받음  
>>> prime  
[2, 3, 5]  
>>> a  
7
```

다음과 같이 리스트의 원소에 새로운 값을 지정할 수도 있습니다.

```
>>> prime[0] = 1  
>>> prime  
[1, 3, 5]
```

리스트에 리스트를 집어넣을 수도 있지요. 피자가게에서 주문할 음식 리스트를 볼까요?

```
>>> orders = ['potato', ['pizza', 'Coke', 'salad'], 'hamburger']  
>>> orders[1]  
['pizza', 'Coke', 'salad']  
>>> orders[1][2]  
'salad'
```

마찬가지로 리스트를 사용해서 간단히 행렬을 표현할 수도 있습니다. 행렬은 아마 고등학교 때 배우죠?

```
>>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

문자열을 리스트로 바꾸기

지금까지 문자열과 리스트를 따로따로 알아봤는데요, 이번엔 둘 다 갖고 놀아봅시다. 문자열을 리스트로 바꿔보도록 하죠.

```
>>> characters = []
>>> sentence = 'Be happy!'
>>> for char in sentence:
...     characters.append(char)
...
>>> print(characters)
['B', 'e', ' ', 'h', 'a', 'p', 'p', 'y', '!']
```

처음에 `characters`라는, 비어있는 리스트를 만들었습니다. 그리고, `sentence`라는 변수를 만들어서 `Be happy!`라는 문자열을 가리키도록 했지요. 전에 `for` 문을 배울 때에는 리스트를 이용해서 이터레이션을 수행하였는데, 이번에는 문자열을 이용해보았습니다. 여기선 `sentence`가 가리키는 `Be happy!`의 글자 하나하나에 대해서 어떤 일을 수행하게 되죠. 첫번째 글자인 `B`를 `characters`라는 리스트의 첫번째 원소로 넣고, 두번째 글자인 `e`를 `characters`의 두번째 원소로 넣는 식입니다.

사실은 아래처럼 문자열을 바로 리스트로 변환해도 같은 결과를 얻을 수 있답니다.

```
>>> list('Be happy!')
['B', 'e', ' ', 'h', 'a', 'p', 'p', 'y', '!']
```

숫자를 문자열로 바꾸기

정수 (`int`) 123을 가리키는 `my_int` 변수가 있다고 합시다.

```
>>> my_int = 123
>>> type(my_int)
<class 'int'>
```

문자열 '`123`'을 얻고 싶다면 다음과 같이 할 수 있습니다.

```
>>> str(my_int)
'123'
```

위 출력 결과를 보시면 작은따옴표가 붙어 있습니다. 타입을 확인해보면 더 정확히 알 수 있죠.

```
>>> type(str(my_int))
<class 'str'>
```

이렇게 얻은 문자열을 다음과 같이 변수에 할당하는 것도 가능합니다.

```
>>> my_str = str(my_int)
```

엄밀히 말하자면 ‘숫자를 문자열로 바꾼’ 것이라기보다는, ‘숫자값을 가지고 새로운 문자열을 얻었다’고 표현하는 것이 맞겠죠.

문자열을 숫자로 바꾸기

역으로, 숫자를 나타낸 문자열에서 숫자를 얻어낼 수도 있습니다.

```
>>> int('123')  
123  
>>> float('123')  
123.0
```

리스트 원소들의 합 구하기

1부터 10 까지의 정수를 원소로 갖는 리스트 one_to_ten이 있습니다.

```
>>> one_to_ten = list(range(1, 11))  
>>> one_to_ten  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

1부터 10 까지를 더한 값은 얼마일까요?

for 문을 사용해서 계산할 수도 있겠지만, `sum()`을 이용하면 손쉽게 구할 수 있답니다.

```
>>> sum(one_to_ten)  
55
```

성적표

오늘의 종합편! 우리 반 성적표를 만들어 봅시다! 학생 이름에 국, 영, 수 성적을 넣어주고,

```
>>> chulsu = [90, 85, 70]  
>>> younghee = [88, 79, 92]
```

```
>>> yong = [100, 100, 100] # 바로 접니다..
>>> minsu = [90, 60, 70]
```

우리 반 학생들을 전부 `students` 리스트에 넣어줍니다.

```
>>> students = [chulsu, younghee, yong, minsu]
```

학생들의 성적이 어떤지 불러내볼까요?

```
>>> for scores in students:
...     print(scores)
...
[90, 85, 70]
[88, 79, 92]
[100, 100, 100]
[90, 60, 70]
```

개인의 성적을 더해서 총점, 평균도 내 보세요.

```
>>> for scores in students:
...     total = 0
...     for s in scores:
...         total = total + s
...     average = total / 3
...     print(scores, total, average)
...
[90, 85, 70] 245 81.66666666666667
[88, 79, 92] 259 86.33333333333333
[100, 100, 100] 300 100.0
[90, 60, 70] 220 73.33333333333333
```

위에서는 총점을 구하기 위해 `total`이라는 변수를 사용해서 숫자를 누적시키는 방법을 써보았습니다.

그럼, 오늘 강좌 끝 ~.

4.2.1 연습 문제: 각 자리 숫자의 합을 구하는 함수 (리스트를 이용)

문제

정수 `num`을 매개변수로 받아 각 자리 숫자 (digit)의 합을 계산하는 `sumOfDigits()` 함수를 작성하세요. 단, 나눗셈을 이용하지 말고 리스트를 사용해서 풀어보세요.

예 1

입력:

```
643
```

출력 ($6 + 4 + 3 = 13$):

```
13
```

예 2

입력:

```
47253
```

출력:

```
21
```

힌트

정수의 나눗셈과 나머지 구하는 법: [1.2 숫자 계산](#)을 보세요.

사용자 입력을 받는 방법: [1.5. 명령해석기](#)를 보세요.

답

- 코드: [ch04/sumOfDigits_non-recursive_list.py](#)

4.2.2 연습 문제: 줄기와 잎 그림

중학교 1 학년 수학 시간에 배우는 줄기와 잎 그림을 파이썬으로 만들어볼게요. 배운 지 오래 돼서 기억이 나지 않거나 아직 배우지 않았더라도 걱정하지 않으셔도 됩니다. 칸 아카데미의 [줄기와 잎 그림 수업](#)을 들어보면 금방 이해하실 거예요.

문제

농구팀에 속한 12 명의 득점을 다음과 같이 `score` 리스트로 나타냈습니다.

```
>>> score = [0, 0, 2, 4, 7, 7, 9]
>>> score += [11, 11, 13, 18]
>>> score += [20]
>>> score
[0, 0, 2, 4, 7, 7, 9, 11, 11, 13, 18, 20]
```

다음과 같이 `stem_leaf` 리스트를 만들어 선수들의 득점을 줄기와 잎 그림 형식으로 저장하려고합니다.

```
>>> stem_leaf = [[], [], []]
```

문제 1

`stem_leaf`에 데이터를 채우는 프로그램을 작성하세요. 데이터가 채워진 결과는 다음과 같습니다.

```
>>> stem_leaf
[[0, 0, 2, 4, 7, 7, 9], [1, 1, 3, 8], [0]]
>>> stem_leaf[0]
[0, 0, 2, 4, 7, 7, 9]
>>> stem_leaf[1]
[1, 1, 3, 8]
>>> stem_leaf[2]
```

[0]

문제 2

`stem_leaf`를 다음과 같은 형태로 프린트하는 프로그램을 작성하세요.

```
0: [0, 0, 2, 4, 7, 7, 9]
1: [1, 1, 3, 8]
2: [0]
```

답

- 코드: [ch04/stem_leaf.py](#)

4.2.3 연습 문제: 각 자리 숫자의 합을 구하는 함수 (`map()` 을 이용)

문제

정수 `num`을 매개변수로 받아 각 자리 숫자 (digit) 의 합을 계산하는 `sumOfDigits()` 함수를 작성하세요. 단, 나눗셈을 이용하지 말고, 리스트와 `map()` 함수를 이용해 풀어보세요.

예 1

입력:

```
47253
```

출력:

```
21
```

예 2

입력:

```
643
```

출력:

```
13
```

답

- 코드: [ch04/sumOfDigits_non-recursive_map.py](#)

4.2.4 연습 문제: 소수 구하기

문제

소수 (素數, prime number) 는 1 과 그 자체만을 인수 (factor) 로 갖는 수입니다¹. 또는 “1 보다 큰 자연수 중 1 과 자기 자신만을 약수로 가지는 수” 라고 설명하기도 합니다².

다음은 소수입니다.

```
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, ...
```

소수를 구하는 방법은 다음과 같습니다.

1. 찾고자 하는 범위의 자연수를 나열한다.
2. 2부터 시작하여, 2의 배수를 지워나간다.
3. 다음 소수의 배수를 모두 지운다.

다음은 파이썬 셀에서 수작업으로 10 이하의 소수를 구하는 예입니다.

```
>>> L = list(range(2, 11))          # 찾고자 하는 범위의 자연수를 나열
>>> L
[2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> L.remove(4); L.remove(6); L.remove(8); L.remove(10) # 2의 배수를 지움
>>> L.remove(9)                      # 3의 배수를 지움
>>> L                                # 결과
[2, 3, 5, 7]
```

이와 같은 방식으로 찾고자 하는 범위 (예: 30 이하) 의 소수를 구하는 것이 문제입니다. 수작업으로 하지 말고 반복문을 사용하세요.

¹칸 아카데미, Prime Numbers

²https://ko.wikipedia.org/wiki/%EB%8A%A8_%EB%8A%A8%28%EB%8A%A8%29

예

예 1

입력:

```
10
```

출력:

```
[2, 3, 5, 7]
```

예 2

입력:

```
20
```

출력:

```
[2, 3, 5, 7, 11, 13, 17, 19]
```

예 3

입력:

```
30
```

출력:

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

답

첫 번째 풀이

처음에는 리스트의 `remove()` 메서드를 이용해 간단하게 풀고, 그 과정을 유튜브 동영상으로 올려두었습니다.

- <https://youtu.be/KChXYFu2rYo>

그런데 이 방법이 이해하기는 쉽지만 몇 가지 문제점이 있다는 것을 나중에 발견했습니다. 리스트를 가지고 `for` 루프를 수행하는 도중에 리스트의 원소를 삭제하는 바람에 모든 원소를 제대로 검사하지 않는다는 것이었습니다. 위의 동영상은 그 문제를 알지 못하고 찍은 것입니다.

그 문제를 해결하기 위해, 다음과 같이 리스트를 복사해서 사용하도록 수정했습니다.

```
L2 = L[:]
```

아래 주소의 코드는 위의 변경사항이 반영된 것입니다.

- [ch04/prime.py](#)

그런데 이 코드에는 또 다른 문제점이 있습니다. 2의 배수를 모두 검사하고 나서 3의 배수를 검사할 때는 숫자 2는 확인할 필요가 없는데, 그 점을 미처 생각지 못하고 모두 검사하게 되어 있어 비효율적입니다.

두 번째 풀이

첫 번째 풀이의 문제점들을 해결한 코드를 새로 작성했습니다. `for` 루프 대신 `while` 루프를 사용해서 첫 번째 문제점을 해결했고, `while` 루프의 조건식을 이용해 불필요한 검사를 하지 않게 해 효율을 높였습니다.

- [ch04/prime2.py](#)

전보다 나아지긴 했지만, 다른 책에 실린 코드와 비교해보니 복잡하면서 비효율적인 부분이 여전히 남아 있는 것 같습니다.

독자분들의 풀이

독자분들께서 제안해주신 풀이입니다. 고맙습니다!

- 김 님의 풀이 (**if** 절을 추가): [ch04/prime_kim.py](#)
- fateindestiny.dev 님의 풀이 (**filter**를 활용): [ch04/prime_fate.py](#)
- aaaa 님의 풀이: [ch04/prime_aaaa.py](#)

프로그램 실행 시간 비교

여러 방법으로 구현한 코드의 성능을 [프로그램 실행 시간 측정하기](#)에서 비교합니다.

더 읽을 거리

위키독스에 소수 구하기를 다룬 책들이 있습니다. 풀이 방법을 비교해 보세요.

- 김중운, [파이썬 계단 밟기](#)
- 이연홍, [코딩으로 수학하기](#)
- Kyoungwon, [중학수학 코딩의 정석](#)

4.2.5 연습 문제: 진법 변환

나눗셈을 이용해 십진수를 이진수로 변환하는 방법은 다음과 같습니다.

십진수를 2로 나눈 몫을 구하고 그 수를 다시 2로 나누는 일을 몫이 0이 될 때까지 반복하고, 각 단계에서 구한 나머지를 거꾸로 쓰면 이진수가 됩니다.

예를 들어, 십진수 13에 해당하는 이진수를 구하는 과정을 아래와 같이 나타낼 수 있습니다 ([몫과 나머지](#)를 한 번에 구하기 참조).

```
>>> divmod(13, 2) # 13을 2로 나눈 몫은 6, 나머지는 1  
(6, 1)  
>>> divmod(6, 2) # 6을 2로 나눈 몫은 3, 나머지는 0  
(3, 0)  
>>> divmod(3, 2) # 3을 2로 나눈 몫은 1, 나머지도 1  
(1, 1)  
>>> divmod(1, 2) # 1을 2로 나눈 몫은 0, 나머지는 1  
(0, 1)
```

위의 각 단계에서 구한 나머지 1, 0, 1, 1을 역순으로 쓴 1101이 십진수 13에 해당하는 이진수입니다. 파이썬의 [bin\(\)](#) 함수를 이용하면 위와 같이 번거로운 계산을 직접 하지 않고도 이진수를 쉽게 구할 수 있죠.

```
>>> bin(13)  
'0b1101'
```

0b는 뒤의 숫자가 이진수임을 나타냅니다.

문제

십진수를 입력받아 그 숫자에 해당하는 이진수의 각 자리를 리스트로 출력하는 프로그램을 작성하세요.
(순서에 유의)

예 1

입력:

```
13
```

출력:

```
[1, 1, 0, 1]
```

예 2

입력:

```
87
```

출력:

```
[1, 0, 1, 0, 1, 1, 1]
```

답

- 코드: [ch04/dec2bin.py](#)

4.3 튜플 (tuple)

안녕하세요! 오늘 날씨가 아주 좋네요.

출근하지 말고 어디 놀러가고 싶더라고요. 강좌를 쓰는 것이 시간이 많이 걸리고 가끔 머리 아프긴 해도 너무 재미있네요. 그래서 요즘은 틈만 나면 여기에 매달려있답니다.

오늘은 튜플이라는 자료형이 어떤 쓸모가 있는지 알아보도록 하죠. 영어의 tuple을 ‘튜플’ 혹은 ‘터플’이라고 읽습니다. 리스트와 비슷한 자료형이라는 정도만 알고 시작해봅시다.

다른 언어를 공부해보신 분은 두 변수의 값을 서로 바꾸어 본 적이 있으실 텐데요, 보통 다음과 같은 방법을 사용합니다.

```
>>> a = 10
>>> b = 20
>>> temp = a                      # a 값을 temp에 저장      (temp = 10)
>>> a = b                          # b 값을 a에 저장      (a = 20)
>>> b = temp                       # temp 값을 b에 저장      (b = 10)
>>> print(a, b)
20 10
```

이렇게 두 변수 값을 맞바꾸기 위해선 또 다른 변수 temp가 필요합니다. 좀 번거롭죠? 변수가 많을수록 더 귀찮아질테구요. 그런데 파이썬에선 이런 일을 간단하게 할 수 있는 기막힌 방법이 있습니다.

```
>>> c = 10
>>> d = 20
>>> c, d = d, c
>>> print(c, d)
20 10
```

너무 간단하지요? 저는 이것을 보고 웃어버렸습니다. 헬嘿嘿...

세 번째 줄에서 등호 왼쪽은 c, d라는 변수가 담긴 튜플이구요, 오른쪽은 d와 c의 값이 담긴 튜플입니다. 그래서 d의 값은 c로 들어가고, c의 값은 d로 들어갑니다. 이런 일들이 차례차례 일어나는 것이 아니고, 동시에 처리된다는군요.

왕초보를 위한 Python: 쉽게 풀어 쓴 기초 문법과 실습

이번엔 함수에서 튜플이 요긴하게 쓰이는 것을 보여드리지요. 아래의 함수는 인자 (매개변수) 를 주는 대로 받아먹는 함수입니다.

```
>>> def magu_print(x, y, *rest):          # 마구 찍어 함수
...     print(x, y, rest)
...
>>> magu_print(1, 2, 3, 5, 6, 7, 9, 10)
1 2 (3, 5, 6, 7, 9, 10)
```

위 함수는 인자를 두 개 이상만 주면 나머진 다 알아서 처리한답니다. 함수를 정의할 때 인자에 별표를 붙여두면 그 이후에 들어오는 것은 모두 튜플에 집어넣는 것이죠. 위에선 (3, 5, 6, 7, 9, 10) 가 하나의 튜플로 묶였습니다. 꽤 쓸만할 것 같죠? 다른 언어로 이런 함수를 만들려면 고생 꽤나 해야 할 거예요. 인자를 두 개, 세 개 넣어서도 실험해 보세요.

튜플의 좋은 점들을 구경했으니 이제 문법을 살펴봅시다.

```
>>> t = ('a', 'b', 'c')
```

튜플을 만들 때는 위와 같이 괄호를 써도 되고 안 써도 됩니다. 다만, 원소가 없는 튜플을 만들 때는 괄호를 꼭 써주세요.

```
>>> empty = ()
```

원소를 하나만 가진 튜플을 만들 땐 원소 뒤에 콤마 (,) 를 꼭 찍어주시구요.

```
>>> one = 5,
>>> one
(5,)
```

그리고 튜플은 리스트와 달리 원소값을 직접 바꿀 수 없기 때문에, 문자열에서 했던 것처럼 오려붙이는 방법을 써야한다는 것을 알아두세요.

```
>>> p = (1,2,3)
>>> q = p[:1] + (5,) + p[2:]
>>> q
(1, 5, 3)
>>> r = p[:1], 5, p[2:]
>>> r
((1,), 5, (3,))
```

튜플을 리스트로, 리스트를 튜플로 쉽게 바꿀 수도 있답니다.

```
>>> p = (1, 2, 3)
>>> q = list(p)                      # 튜플 p로 리스트 q를 만듦
>>> q
[1, 2, 3]
>>> r = tuple(q)                     # 리스트 q로 튜플 r을 만듦
>>> r
(1, 2, 3)
```

그럼, 여러분 안녕~.

4.3.1 연습 문제: 내일의 날짜 구하기 (1)

문제¹

사용자로부터 날짜를 나타내는 세 개의 숫자를 입력받습니다. 첫 번째 숫자는 연도를 나타내는 네 자리 숫자이고, 두 번째 숫자는 월을, 세 번째 숫자는 일을 나타냅니다.

입력받은 날짜를 `mm/dd/yyyy` 형식으로 출력합니다. 월을 두 자리 숫자 (01, 02, 03, ..., 12)로, 일을 두 자리 숫자 (01, 02, 03, ..., 31)로, 연도를 네 자리 숫자로 나타냅니다.

입력받은 날짜의 다음 날에 해당하는 날짜도 같은 형식으로 출력합니다. 단, 윤년은 무시합니다 (2월은 항상 28 일까지 있다고 가정합니다).

예

예 1

입력:

```
2018 10 2
```

출력:

```
10/02/2018  
10/03/2018
```

예 2

입력:

¹edX.org 의 [C Programming: Advanced Data Types](#) 강좌에 나온 연습 문제입니다.

2018 10 31

출력:

10/31/2018
11/01/2018

예 3

입력:

2018 11 30

출력:

11/30/2018
12/01/2018

예 4

입력:

2018 12 31

출력:

12/31/2018
01/01/2019

답

- 코드: [ch04/tomorrow.py](#)

4.4 딕셔너리 (dict)

오늘 제가 여러분과 함께 공부할 것은 딕셔너리 자료형이예요. 사전을 한번도 못 보신 분은 안 계시죠?

dictionary

n. pl. dictionaries

1. A reference book containing an alphabetical list of words, ...

python

n.

Any of various nonvenomous snakes of the family Pythonidae, ...

딕셔너리 자료형으로 꼭 국어사전이나 백과사전 같은 것을 만들어야 하는 건 아니지만, 기억하기 쉽도록 영어 사전을 만들어 보겠습니다. 위의 자료들을 우리가 지금까지 배운 자료형을 사용해서 저장하면 어떤 것이 좋을까요? 저장해뒀다가 dictionary 라고 치면 'A reference book 주절주절...' 하고 나오고, python 이라고 하면 또 '궁시렁궁시렁...' 하도록 말이죠.

리스트?

튜플?

물론 그런 것들을 사용해도 만들 수는 있겠지만 결국 우리가 배울 딕셔너리 자료형과 비슷한 구조를 만들게 될 듯 싶네요. 딕셔너리 자료형은 아래와 같이 사용할 수 있습니다.

```
>>> dic = {}          # dic이라는 이름으로 비어 있는 딕셔너리를 만든다.  
>>> dic['dictionary'] = '1. A reference book containing an ...'  
>>> dic['python'] = 'Any of various nonvenomous snakes of the ...'  
>>> dic['dictionary']      # dic아, 'dictionary'가 뭐니?  
'1. A reference book containing an ...'
```

처음에 dic이라는 사전을 하나 만들고, 둘째, 셋째 줄에서는 dic에다가 자료를 좀 집어넣었지요. 그리고, 마지막 줄에선 dictionary의 뜻이 뭔지 조회를 해봤습니다.

영어가 너무 많아서 겁이 나십니까? ;-; 프로그래머가 영어 겁내서 쓰나요. 앞으로 공부를 하면 할수록 영어의 중요성을 느끼시게 될 겁니다.

그래도 영어에 약하신 분을 위해 포켓용 사전을 만들어 볼까요?

```
>>> smalldic = {'dictionary': 'reference', 'python': 'snake'}  
>>> smalldic['python']          # 포켓용 사전아, 'python'이 뭐니??  
'snake'  
>>> smalldic  
{'dictionary': 'reference', 'python': 'snake'}
```

좀 더 깔끔해졌죠? 유심히 보시면 아까와는 조금 다른 방법으로 딕셔너리를 만들었다는 것도 아실 수 있겠지요?

이와 같이 딕셔너리 자료형은 키 (key) 와 값 (value) 의 쌍으로 이루어진답니다. 아래의 표를 참고하세요.

| 키 | 값 |
|------------|-----------|
| dictionary | 주절주절... |
| python | 궁시렁궁시렁... |
| zoo | 동물원 |

문자열, 리스트, 튜플은 숫자로 된 인덱스를 이용해 값을 조회하는데, 딕셔너리는 키를 이용한다는 것이 큰 차이점이죠. 또, 딕셔너리 자료형은 해싱 (hashing) 기법을 이용하기 때문에 자료가 순서대로 저장되지 않는다고 하네요.

해싱 기법이 무엇일까요? 선생님이 학생들의 시험지를 보니 만득이가 빵점을 받았다고 합시다. 그래서 만득이를 찾아서 혼내주려고 하는데,

“1 번, 네가 만득이니?” “아니요.”

“2 번, 네가 만득이니?” “아니오.”

“3 번, 네가 만득이니?” “아니오.”

...

“40 번, 네가 만득이니?” “아니오.”

이렇게 만득이를 찾으려면 좀 오래 걸리겠죠? 하지만, 선생님이 “만득이 나와!”라고 하시면 바로 찾을 수 있지 않겠습니까? 이 방법이 바로 해싱 기법과 비슷하다고 생각하시면 됩니다. 자료를 아주 빨리 찾을 수 있는 방법이지요.

딕셔너리 자료형을 만들고, 원소를 추가하는 방법은 위에서 보신 대로이구요, 원소를 삭제할 땐 이렇게 하시면 됩니다.

```
>>> del smalldic['dictionary']
```

삭제가 잘 되었는지 한번 확인해보세요.

이번에는 `family`라는 딕셔너리를 만들어 볼게요.

```
>>> family = {'mom': 'Kim', 'dad': 'Choi', 'baby': 'Choi'}  
>>> family  
{'mom': 'Kim', 'dad': 'Choi', 'baby': 'Choi'}
```

`family`의 키들을 얻으려면 딕셔너리 이름 뒤에 `.keys()`를 쓰면 됩니다.

```
>>> family.keys()  
dict_keys(['mom', 'dad', 'baby'])
```

`family`의 값들을 얻으려면 딕셔너리 이름 뒤에 `.values()`를 쓰면 됩니다.

```
>>> family.values()  
dict_values(['Kim', 'Choi', 'Choi'])
```

`family`의 원소 (키/값 쌍) 들을 얻으려면 이름 뒤에 `.items()`를 쓰면 됩니다.

```
>>> family.items()  
dict_items([('mom', 'Kim'), ('dad', 'Choi'), ('baby', 'Choi')])
```

딕셔너리에 어떤 키가 있는지 없는지는 `in`을 써서 알아볼 수 있습니다. 있으면 `True`, 없으면 `False`라고 대답해주죠.

```
>>> 'dad' in family  
True  
>>> 'sister' in family  
False
```

이런 것들을 외우려고 애쓰실 필요는 없습니다. 필요할 때마다 사용방법을 찾아서 쓰면 되는거지요.

그럼 모두 즐거운 하루 되세요 ~.

4.4.1 연습 문제: 숫자 읽기 (0~9)

문제

매개변수로 입력받은 정수에 해당하는 한글 문자열을 반환하는 함수 `korean_number()`를 **if 문을 사용하지 말고** 작성하세요. 단, 사용자는 0 이상 9 이하의 정수 중 하나를 입력한다고 가정합니다.

예

```
>>> korean_number(3)
'삼'
>>> korean_number(6)
'육'
>>> korean_number(9)
'구'
```

답

- 코드: [ch04/korean_0_to_9.py](#)

4.4.2 연습 문제: 한자 성어

문제

다음 표의 한자 성어와 뜻을 모두 출력하는 파이썬 프로그램을 작성하세요.¹

| 한자 성어 | 뜻 |
|-------------|-----------------------------|
| 江湖之樂 (강호지락) | 자연을 벗 삼아 누리는 즐거움 |
| 欲速不達 (욕속부달) | 빨리 하고자 하면 이루지 못함 |
| 積小成大 (적소성대) | 작은 것을 쌓아 큰 것을 이룸 |
| 勤儉節約 (근검절약) | 부지런하고 알뜰하게 재물을 아낌 |
| 經世濟民 (경세제민) | 세상을 다스리고 백성을 구제함 |
| 塞翁之馬 (새옹지마) | 인생의 길흉화복은 변화가 많아서 예측하기가 어려움 |
| 好事多魔 (호사다마) | 좋은 일에는 흔히 방해되는 일이 많음 |
| 桑田碧海 (상전벽해) | 세상일의 변천이 심함 |
| 自業自得 (자업자득) | 자기가 저지른 일의 결과를 자기가 받음 |
| 因果應報 (인과응보) | 원인과 결과가 상응하여 보답한다 |
| 愚公移山 (우공이산) | 어떤 일이든 끊임없이 노력하면 반드시 이루어짐 |

실행 결과:

```
Enter를 누르세요...
江湖之樂(강호지락)
```

¹출처: 서울사이버대학교 ‘한자성어와퍼즐’ 수업

자연을 벗 삼아 누리는 즐거움

Enter를 누르세요...

欲速不達(욕속부달)

빨리 하고자 하면 이루지 못함

Enter를 누르세요...

積小成大(적소성대)

작은 것을 쌓아 큰 것을 이룸

(이하 생략)

풀이

- 코드: ch04/hanja_idioms.py

4.4.3 연습 문제: 정신 질환

문제

다음 `txt`는 정신질환의 명칭을 한국어와 영어 용어로 나열한 것입니다.¹

```
txt = '''신경 발달 장애 Neurodevelopmental Disorders  
조현병 스펙트럼 및 기타 정신병적 장애 Schizophrenia Spectrum and Other Psychotic  
Disorders  
양극성 및 관련 장애 Bipolar and Related Disorders  
우울장애 Depressive Disorders  
불안장애 Anxiety Disorder  
강박 및 관련 장애 Obsessive—Compulsive and Related Disorders  
외상 및 스트레스 관련 장애 Trauma—and Stressor—Related Disorders  
해리장애 Dissociative Disorders  
신체증상 및 관련 장애 Somatic Symptom and Related Disorders  
급식 및 섭식장애 Feeding and Eating Disorders  
배설장애 Elimination Disorders  
수면—각성 장애 Sleep—Wake Disorders  
성기능부전 Sexual Dysfunctions  
성별 불쾌감 Gender Dysphoria  
파괴적, 충동조절 및 품행 장애 Disruptive, Impulse—Control, and Conduct Disorders  
물질관련 및 중독 장애 Substance—Related and Addictive Disorders  
신경인지장애 Neurocognitive Disorders  
성격장애 Personality Disorders  
변태 성욕장애 Paraphilic Disorders  
기타 정신질환 Other Mental Disorders'''
```

`txt`를 읽어 한국어와 영어 명칭을 각각 키와 값으로 갖는 딕셔너리를 생성하는 코드를 작성하세요.

결과

```
{'신경 발달 장애': 'Neurodevelopmental Disorders', '조현병 스펙트럼 및 기타 정신병적  
장애': 'Schizophrenia Spectrum and Other Psychotic Disorders', '양극성 및 관  
련 장애': 'Bipolar and Related Disorders', '우울장애': 'Depressive Disorders',  
'불안장애': 'Anxiety Disorder', '강박 및 관련 장애': 'Obsessive—Compulsive
```

¹APA 저/권준수, 김재진, 남궁기, 박원명 역, [정신질환의 진단 및 통계편람](#), 학지사

```
and Related Disorders', '외상 및 스트레스 관련 장애': 'Trauma—and Stressor—Related Disorders', '해리장애': 'Dissociative Disorders', '신체증상 및 관련 장애': 'Somatic Symptom and Related Disorders', '급식 및 섭식장애': 'Feeding and Eating Disorders', '배설장애': 'Elimination Disorders', '수면—각성 장애': 'Sleep—Wake Disorders', '성기능부전': 'Sexual Dysfunctions', '성별 불쾌감': 'Gender Dysphoria', '파괴적, 충동조절 및 품행 장애': 'Disruptive, Impulse—Control, and Conduct Disorders', '물질관련 및 중독 장애': 'Substance—Related and Addictive Disorders', '신경인지장애': 'Neurocognitive Disorders', '성격장애': 'Personality Disorders', '변태 성욕장애': 'Paraphilic Disorders', '기타 정신질환': 'Other Mental Disorders'}
```

ord() 와 chr()

`ord()` 함수는 문자에 해당하는 코드값을 알려줍니다.

```
>>> ord('A')
65
>>> ord('Z')
90
>>> ord('a')
97
>>> ord('z')
122
>>> ord('0')
48
>>> ord('9')
57
```

역으로, `chr()` 함수에 코드값을 입력으로 넣으면 그에 해당하는 문자를 얻습니다.

```
>>> chr(65)
'A'
```

한글에 대해서도 두 함수를 사용할 수 있습니다.

```
>>> ord('가')
44032
>>> chr(55197)
'힝'
```

split() 과 splitlines()

split()은 공백을 기준으로 문자열을 분리한 것들을 리스트에 넣어 반환합니다.

```
>>> 'hello world'.split()  
['hello', 'world']
```

여러 행으로 이뤄진 문자열을 분리하려면 \n을 구분자로 지정합니다.

```
>>> love = '''L is for the way you look at me  
... O is for the only one I see  
... V is very, very extraordinary  
... E is even more than anyone that you adore can'''  
>>> love.split('\n')  
['L is for the way you look at me', 'O is for the only one I see', 'V is  
very, very extraordinary', 'E is even more than anyone that you adore can  
'']
```

위와 같이 줄바꿈을 기준으로 문자열을 분할할 때는 splitlines()를 써도 됩니다.

답

- 코드: ch04/mental_disorders.py

4.4.4 연습 문제: 프랙털

0과 1로 이루어진 문자열 한 행이 있을 때, 정해진 규칙에 따라 그다음 행을 생성하는 문제입니다.

다음과 같이 이어진 세 숫자 조합에 따라 다음에 올 숫자가 정해집니다.

| 현재 패턴 | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
|-----------------|-----|-----|-----|-----|-----|-----|-----|-----|
| 가운데 자리에 새로 올 숫자 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |

예를 들어, 첫 행이 000010000이면 그다음 행은 000101000이 됩니다.

000010000
000101000

다음은 한 행에 61 개의 숫자가 있는 예입니다. 첫 번째 행은 가운데 숫자가 1, 나머지는 모두 0입니다. 두 번째 행부터는 규칙에 따라 생성합니다.

답

- 코드: [ch04/rule90.py](#)

참고

- Rule 90
 - 《Nature of Code》

4.5 세트 (set)

이번에는 ‘집합’을 표현하는 **세트 (set)**를 좀 더 알아보겠습니다.

과일을 나타내는 `fruits` 세트를 만들어보겠습니다남 ~

```
>>> fruits = {'apple', 'banana', 'orange'}
```

사과, 바나나, 오렌지를 원소로 갖는 `fruits` 세트를 만들었습니다. 이와 같이 세트는 중괄호 (`{, }`)를 사용합니다.

아차, 맛있는 망고를 빠뜨렸네요. `add()`로 추가할게요.

```
>>> fruits.add('mango')
>>> fruits
{'orange', 'apple', 'mango', 'banana'}
```

이번에는 회사 이름을 나타내는 집합을 만들어볼까요?

```
>>> companies = set()
```

회사 이름이 떠오르지 않아서 일단 `set()`로 빈 세트를 만들었습니다. 아, 생각 났어요.

```
>>> companies = {'apple', 'microsoft', 'google'}
```

이제 `fruits`와 `companies` 세트를 만들었습니다. 타입을 확인해볼까요?

```
>>> type(fruits)
<class 'set'>
>>> type(companies)
<class 'set'>
```

세트를 이용해 아래와 같이 집합 연산을 사용할 수 있습니다.

```
>>> fruits & companies          # 교집합  
{'apple'}  
>>> fruits | companies        # 합집합  
{'apple', 'mango', 'microsoft', 'orange', 'google', 'banana'}
```

아래와 같이 여러 세트를 리스트에 담은 뒤 set의 메서드를 쓸 수도 있습니다.

```
>>> list_of_sets = [fruits, companies]  
>>> set.intersection(*list_of_sets)  # 교집합  
{'apple'}  
>>> set.union(*list_of_sets)       # 합집합  
{'google', 'apple', 'banana', 'mango', 'microsoft', 'orange'}
```

apple은 fruits에도 속하고 companies에도 속하는데, 위 합집합의 결과에 한 번만 나오는 것을 볼 수 있습니다. 이와 같이 세트는 중복 원소를 갖지 않습니다. 또, 원소의 순서가 유지되지 않는 특징도 있습니다.

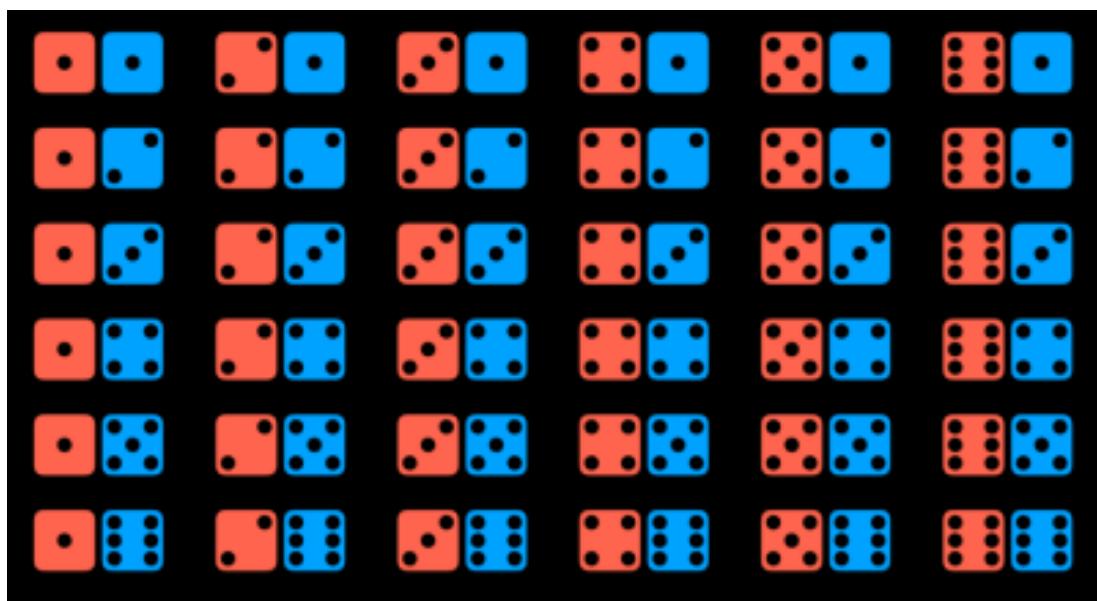
```
>>> alphabet = list('google')  
>>> alphabet  
['g', 'o', 'o', 'g', 'l', 'e']  
>>> set(alphabet)  
{'e', 'o', 'g', 'l'}
```

아참, 집합끼리 뺄셈도 할 수 있어요!

```
>>> S1 = {1, 2, 3, 4, 5, 6, 7}  
>>> S2 = {3, 6, 9}  
>>> S1 - S2  
{1, 2, 4, 5, 7}
```

4.5.1 연습 문제: 주사위 눈의 합

다음 그림은 주사위 두 개를 던져서 나오는 경우의 수를 모두 나열한 것입니다.



(그림 출처: <https://cs50.harvard.edu/ai/2020/notes/2/>)

두 주사위 눈의 합은 다음과 같습니다.

| | | | | | |
|---|---|---|----|----|----|
| 2 | 3 | 4 | 5 | 6 | 7 |
| 3 | 4 | 5 | 6 | 7 | 8 |
| 4 | 5 | 6 | 7 | 8 | 9 |
| 5 | 6 | 7 | 8 | 9 | 10 |
| 6 | 7 | 8 | 9 | 10 | 11 |
| 7 | 8 | 9 | 10 | 11 | 12 |

(그림 출처: <https://cs50.harvard.edu/ai/2020/notes/2/>)

문제

주사위 두 개가 있습니다. 한 개는 평범한 주사위인데, 다른 한 개의 각 면에는 2에서 13까지의 소수가 적혀 있습니다. 아래 코드는 두 주사위의 눈을 튜플 `dice1`과 `dice2`로 나타냅니다.

```
dice1 = (1, 2, 3, 4, 5, 6)
dice2 = (2, 3, 5, 7, 11, 13)
```

두 주사위를 던졌을 때 눈의 합으로 나올 수 있는 숫자를 모두 출력하세요. 단, 같은 숫자는 한 번만 출력합니다.

답

- 코드: [ch04/sum_dice.py](#)

4.5.2 연습 문제: 끝말 잇기 (1)

끝말 잇기를 하는 프로그램을 작성하세요.

컴퓨터와 플레이어는 자기 턴 (turn, 차례) 이 되면 이전에 상대방이 말한 단어의 끝 글자로 시작하는 단어를 말해야 하며, 이전에 썼던 단어는 말할 수 없습니다.

1. 컴퓨터가 먼저 ‘기차’를 출력합니다.
2. 사용자의 입력을 받습니다. 사용자가 올바로 입력했으면 다음으로 진행하고, 단어를 잘못 입력하거나 ‘쳤어’라고 입력하면 컴퓨터가 이기고 끝냅니다.
3. 컴퓨터는 다음의 단어 중 하나를 골라 출력합니다. 마땅한 것이 없으면 사용자가 이기고 끝냅니다.

게 맛 살, 구멍, 글라이더, 기차, 대룡, 더치페이, 롱다리, 리본, 명계, 박쥐, 본넷, 빨대, 살구, 양심, 이빨, 이자, 자율, 주기, 쥐구멍, 차박, 트라이앵글

4. 2~3 을 반복

» 두음 법칙은 무시합니다.

예

예 1

| 턴 | 입출력 |
|------|-------------------------|
| 컴퓨터 | 끝말잇기를 하자. 내가 먼저 말할게. 기차 |
| 플레이어 | 자동차 |
| 컴퓨터 | 글자가 안 이어져. 내가 이겼다! |

예 2

| 턴 | 입출력 |
|------|-------------------------|
| 컴퓨터 | 끝말잇기를 하자. 내가 먼저 말할게. 기차 |
| 플레이어 | 차박 |
| 컴퓨터 | 박쥐 |
| 플레이어 | 쥐구멍 |
| 컴퓨터 | 멍게 |
| 플레이어 | 게시판 |
| 컴퓨터 | 모르겠다. 내가 졌어. |

예 3

| 턴 | 입출력 |
|------|-------------------------|
| 컴퓨터 | 끝말잇기를 하자. 내가 먼저 말할게. 기차 |
| 플레이어 | 차주 |
| 컴퓨터 | 주기 |
| 플레이어 | 기차 |
| 컴퓨터 | 아까 했던 말이야. 내가 이겼어! |

예 4

| 턴 | 입출력 |
|------|-------------------------|
| 컴퓨터 | 끝말잇기를 하자. 내가 먼저 말할게. 기차 |
| 플레이어 | 차량 |

턴 입출력

컴퓨터 모르겠다. 내가 졌어.

풀이

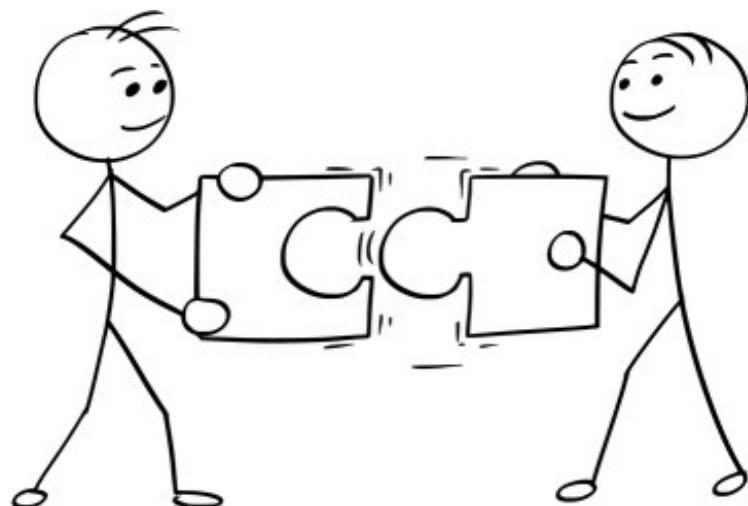
- 코드: [ch04/wordgame.py](#)

5. 모듈

다른 사람이 만들어놓은 모듈 (module) 을 잘 활용하면 쉽고 빠르게 프로그램을 개발할 수 있어요!

이 장에서 배우는 것:

- 모듈이란
- 모듈 가져오기
- 여러 가지 모듈



5.1 모듈이란

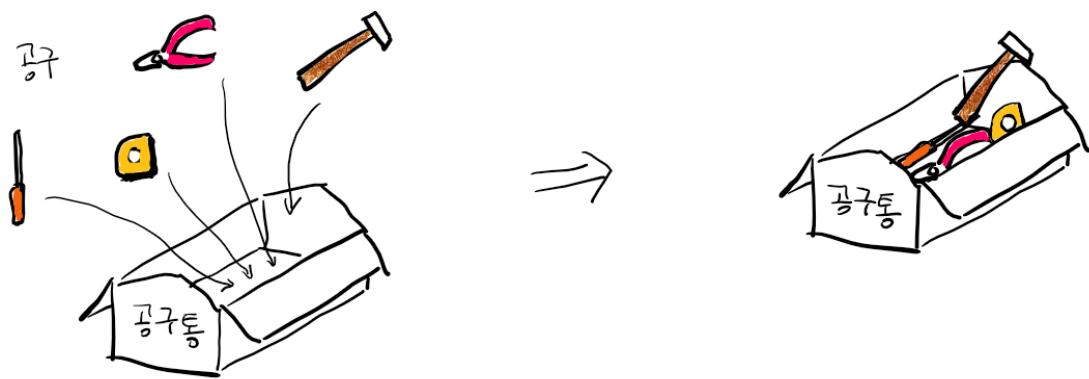
자료구조 부분이 드디어 끝났죠? 좀 지루하셨을 수도 있지만, 컴퓨터에 있어서 자료구조의 중요성은 절대적이라고 할 만큼 크답니다. 작게는 CPU 내의 기억장소에서부터, 크게는 파일, 데이터베이스, 전체 시스템에까지 두루 적용된다고 하니까 틈틈이 공부해주시면 좋겠네요.

우리가 지금까지는 혼자서 변수, 함수를 만들어 쓰면서 자급자족하는 방법을 배웠다고 한다면, 이제부터는 남이 만들어 놓은 부품을 가져다가 사용하는 방법을 배울 차례입니다.

우리가 복잡한 프로그램을 작성하기 위해서 필요한 모든 과정을 직접 만들어야 한다면 어떤 모습이 될까요? 전체적인 모습에서부터 작은 기능 하나하나까지 모두 구상해서, 만들고, 오류를 수정해서 한 곳에 모아두면 또 오류가 생기고... 더구나, 또 다른 프로그래머는 나와 비슷한 기능을 하는 프로그램을 만들면서 똑같은 시행착오를 답습할테구요.

그래서, 이런 문제를 해결하기 위해 대부분의 프로그래밍 언어에서는 모듈이라는 개념을 사용합니다. 모듈은 프로그램의 꾸러미라고 생각하시면 되지요.

할머니 댁 집수리를 해야 해서 여러 가지 공구를 들고 간다고 상상해 봅시다. 손에 다 들고 가기는 불편하니 공구통에 넣는 게 좋겠죠?



math 모듈

왕초보를 위한 Python: 쉽게 풀어 쓴 기초 문법과 실습

수학적인 계산 기능이 필요하다면 math라는 모듈을 불러와서 사용하시면 됩니다.

```
>>> import math # math 모듈을 가져와라
```

제곱근 (square root) 을 구해볼까요?

```
>>> math.sqrt(2) # 2의 제곱근  
1.4142135623730951  
>>> math.sqrt(3) # 3의 제곱근  
1.7320508075688772  
>>> math.sqrt(4) # 4의 제곱근  
2.0
```

원주율도 알아보겠습니다.

```
>>> math.pi # math 모듈의 변수 pi의 값은?  
3.1415926535897931
```

위에서는 math 모듈 내에 정의되어 있는 pi 변수를 사용했습니다. pi는 원주율을 뜻하지요.

calendar 모듈

이번에는 달력을 불러볼까요? 딱 두 줄만 치면 됩니다.

```
>>> import calendar  
>>> calendar.pmonth(2013, 7)  
July 2013  
Mo Tu We Th Fr Sa Su  
1 2 3 4 5 6 7  
8 9 10 11 12 13 14  
15 16 17 18 19 20 21  
22 23 24 25 26 27 28  
29 30 31
```

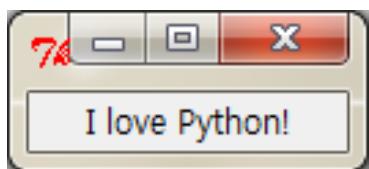
훌륭하죠?

tkinter 모듈

이번에는 더욱 훌륭한 것을 보여드리지요.

```
>>> from tkinter import *  
>>> widget = Label(None, text='I love Python!')
```

```
>>> widget.pack()
```



이렇게 파이썬에서는 좋은 기능들을 모듈로 묶어서 자체적으로 제공해 준답니다. 파이썬 뿐만이 아니라 대부분의 언어에서 이런 식으로 프로그래밍을 편리하게 할 수 있도록 지원해주지요.

모듈을 잘 들여다 보면 배울 것이 많을 것 같군요. 내일 또 만나요 ~.

5.2 모듈 가져오기 (**import**)

어떤 프로그래머께서 이렇게 말씀하셨습니다.

단기간에 뛰어난 프로그래머가 되려고 하면 절대 성공할 수 없느니라.

오늘은 모듈을 어떻게 불러오는지 알아보도록 하죠. 어제 해보셔서 대충은 알고 계시겠지만 **import**를 사용하면 모듈을 불러올 수 있습니다. **import**는 ‘수입하다’, ‘가져오다’라는 뜻을 갖고 있구요, 컴퓨터에서는 다른 프로그램으로부터 데이터를 갖고 오는 것을 뜻하지요.

파이썬에서 임포트를 하는 방법 두 가지를 알아보겠습니다.

첫 번째 방법:

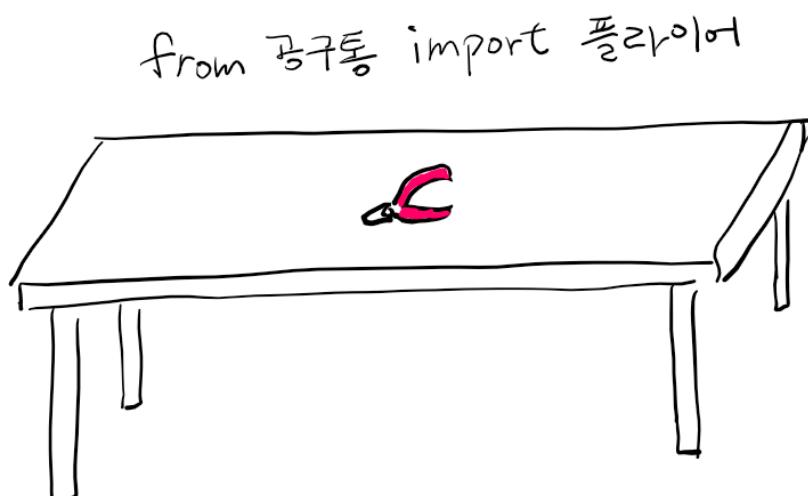
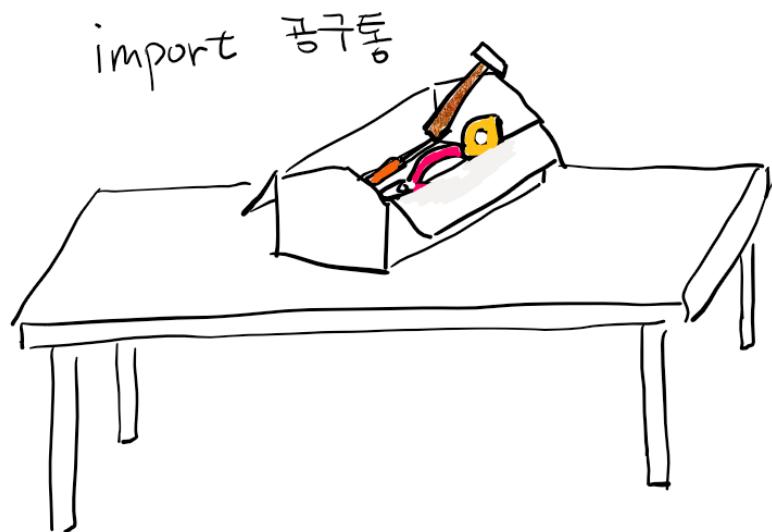
```
import 모듈
```

두 번째 방법:

```
from 모듈 import 이름
```

첫 번째 방법은 모듈 전체를 가져오구요, 두 번째 방법은 모듈 내에서 필요한 것만 쪽 찍어서 가져오는 방법이죠.

공구통에 비유하면 다음 그림과 같이 표현할 수 있어요.



두 방법을 비교해볼까요? 어제 소개해드린 tkinter(티 케이 인터) 모듈을 두 가지 방법으로 임포트해보겠습니다.

```
>>> import tkinter  
>>> tkinter.widget = tkinter.Label(None, text='I love Python!')  
>>> tkinter.widget.pack()
```

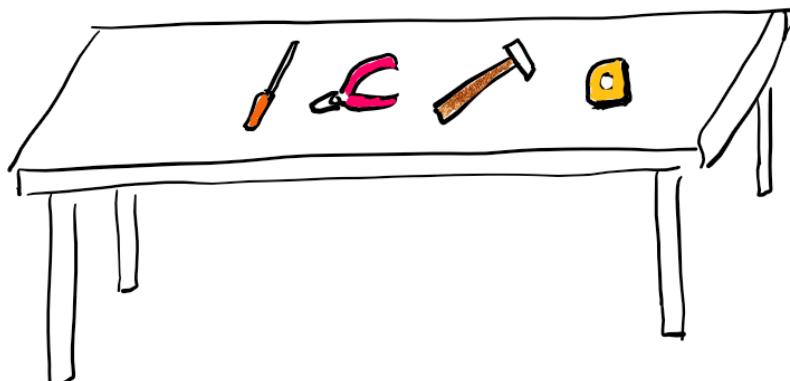
첫 번째 방법으로 모듈을 불러오면 모듈 내의 변수를 사용하기 위해서는 모듈.변수의 형식으로 써주어야

합니다. 매번 써주려면 좀 번거롭겠죠?

```
>>> from tkinter import *
>>> widget = Label(None, text='I love Python!')
>>> widget.pack()
```

두 번째 방법은 모듈 내의 이름을 콕 찍어서 가져오는 방법인데, 위에서는 `tkinter`에 있는 것을 전부 (*) 가져왔습니다. 이렇게 하면 좀 더 편리하군요.

*from 공구통 import **



하지만 마냥 좋기만 한 방법은 아니랍니다. 아래의 예에서는 문자열이었던 `Label`이 임포트 문 실행 후 `tkinter`의 `Label`로 바뀌어 버린 것을 볼 수 있습니다.

```
>>> Label = 'This is a Label'
>>> from tkinter import *
>>> Label
<class 'tkinter.Label'>
```

이런 특성을 이해하고 상황에 맞게 사용하시면 됩니다.

지금까지 모듈을 불러오는 방법을 알아봤는데요, 불러온 모듈이 필요 없을 땐 어떻게 할까요? 필요 없는 모듈은 요렇게 지워주면 됩니다.

del 모듈

꼭 그렇게 해줄 필요가 있을까 싶지만, 프로그램을 짜다보면 이런 저런 일이 생기니까 알아두자구요.

한 번 임포트한 모듈을 다시 불러와야 할 때는 아래와 같이 다시 로드 (reload) 할 수 있답니다.

```
from importlib import reload  
reload(모듈)
```

5.2.1 연습 문제: 모듈 사용법 알아내기

calendar 모듈을 좀 더 살펴보겠습니다.

문제 1. calendar 모듈을 임포트하는 문장을 완성하세요.

```
>>> ---- calendar
```

calendar 모듈에 무엇이 있는지 살펴보세요.

```
>>> dir(calendar)
['Calendar', 'EPOCH', 'FRIDAY', 'February', 'HTMLCalendar', 'IllegalMonthError', 'IllegalWeekdayError', 'January', 'LocaleHTMLCalendar', 'LocaleTextCalendar', 'MONDAY', 'SATURDAY', 'SUNDAY', 'THURSDAY', 'TUESDAY', 'TextCalendar', 'WEDNESDAY', '_EPOCH_ORD', '__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', '_colwidth', '_locale', '_localized_day', '_localized_month', '_monthlen', '_nextmonth', '_prevmonth', '_spacing', 'c', 'calendar', 'datetime', 'day_abbr', 'day_name', 'different_locale', 'error', 'firstweekday', 'format', 'formatstring', 'isleap', 'leapdays', 'main', 'mdays', 'month', 'month_abbr', 'month_name', 'monthcalendar', 'monthrange', 'prcal', 'prmonth', 'prweek', 'repeat', 'setfirstweekday', 'sys', 'timegm', 'week', 'weekday', 'weekheader']
```

이와 같이 모듈은 수많은 공구가 들어 있는 '공구통'이라고 생각할 수 있습니다.

문제 2. calendar 모듈에 leap이라는 문자열을 포함하는 이름은 어떤 것이 있는지 찾아보세요.¹

```
>>> [x for x in dir(calendar) if 'leap' in x]
[----, ----]
```

문제 3. 주어진 해가 윤년인지 아닌지 판별하는 함수의 사용법을 확인해보세요.

```
>>> help(----)
Help on function ---- in module calendar:

----(year)
```

¹이 구문은 [list comprehension](#)이라는 것으로, 'comprehension'은 '컴프리헨션', '내포', '함축', '조건 제시' 등 여러 이름으로 부릅니다.

```
Return True for leap years, False for non-leap years.
```

문제 4. 이 함수를 사용해서 2077년이 윤년인지 아닌지 확인해보세요.

```
>>> ----  
False
```

답

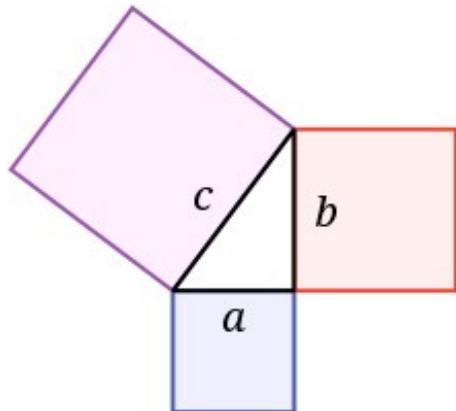
- 코드: [ch05/using_modules.ipynb](#)

5.2.2 연습 문제: 직각삼각형의 빗변 길이 구하기

문제

직각삼각형의 두 직각변 a, b 를 각각 한 변으로 하는 정사각형 면적의 합은 빗변 (hypotenuse) c 를 한 변으로 하는 정사각형의 면적과 같습니다. 이를 ‘피타고라스 정리’라고 합니다. ([위키백과](#))

$$a^2 + b^2 = c^2$$



a 와 b 의 길이를 알면 c 의 길이를 구할 수 있습니다.

$$c = \sqrt{a^2 + b^2}$$

문제 1

$a = 3, b = 4$ 일 때 c 가 얼마인지 파이썬 셸에서 구하세요.

문제 2

두 직각변 (a, b) 의 길이를 입력으로 받아 빗변 (c) 의 길이를 구하는 함수 `hypotenuse()`를 작성하세요. 출력은 소수점 셋째 자리에서 반올림합니다.

```
>>> hypotenuse(3, 4)
5.0
>>> hypotenuse(10, 20)
22.36
```

답

1. [ch05/hypotenuse.txt](#)
2. [ch05/right_triangle.py](#)

참고

- [피타고拉斯 정리 - 칸 아카데미](#)

5.2.3 연습 문제: calendar 와 tkinter

문제

1. calendar

다음은 2021년 2월의 달력을 변수 `m`으로 저장하고 화면에 출력하는 코드입니다. 빈칸을 채우세요.

```
import calendar
c = calendar.TextCalendar()
m = c.██████████(2021, 2)
print(m)
```

결과:

```
February 2021
Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
```

힌트

```
>>> help(c)
```

2. tkinter

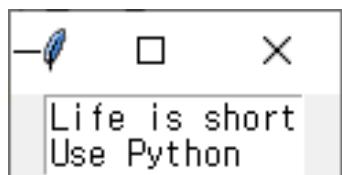
다음은 `tkinter`를 활용해 그림과 같은 창을 띄우는 코드입니다. 빈칸을 채우세요.

```
import tkinter as tk
```

```
s = "Life is short\nUse Python"

root = tk.Tk()
t = tk.Text(root, height=10, width=30)
t.insert(tk.END, s)
t.pack()
tk.mainloop()
```

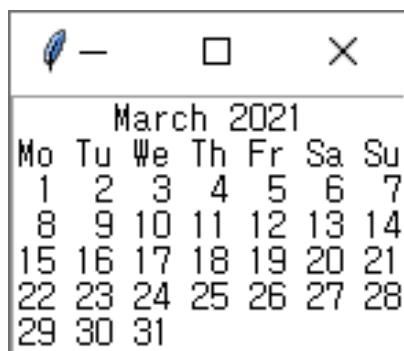
결과:



3. calendar 와 tkinter

그림과 같이 2021년 3월의 달력을 표시하는 창을 띄우는 코드를 작성하세요.

결과:



답

1. [ch05/month2str.py](#)
2. [ch05/life_is_short.py](#)
3. [ch05/tk_calendar.py](#)

5.2.4 연습 문제: 놀이 공원 (2)

둘리, 도우너, 마이콜이 놀이 공원에서 함께 놀이 기구를 타려고 합니다. 그런데 세 명의 키가 달라서, 각자 탈 수 있는 기구가 다릅니다.

문제

이 문제를 풀기 전에 [놀이 공원 \(1\)](#) 문제를 먼저 풀어야 합니다.

다음은 놀이 기구의 목록입니다. 사용자가 키 (신장)를 입력하면, 탈 수 있는 놀이 기구 목록을 출력하는 프로그램을 작성하세요.

```
와일드 윙 : 110cm 이상  
드림보트 : 120cm 이상  
자이안트 루프 : 120cm 이상  
倜 오브 호러 : -  
플라이 벤처 : 140cm~195cm  
회전 목마 : 100cm 이상  
매직 봉봉카 : 110cm~140cm
```

코드는 다음과 같이 작성하며, ch05 폴더에 `park.py`라는 파일명으로 저장합니다.

```
import sys  
  
sys.path.append("../ch03")  
import # 이 곳에 코드를 작성하세요.  
  
rides = # 이 곳에 코드를 작성하세요.  
  
def allowedrides(height):  
    assert type(height) == int  
  
    result = []  
    # 이 곳에 코드를 작성하세요.  
    return result
```

```
if __name__ == "__main__":
    height = int(input())
    # 이곳에 코드를 작성하세요.
```

입출력 예

예 1 둘리의 키를 입력합니다.

```
120
```

출력:

```
와일드윙
드림보트
자이안트루프
툼오브호러
회전목마
매직붕붕카
```

예 2 도우너의 키를 입력합니다.

```
110
```

출력:

```
와일드윙
툼오브호러
회전목마
매직붕붕카
```

예 3 마이콜의 키를 입력합니다.

```
179
```

출력:

```
와일드윙
드림보트
자이안트루프
```

톰 오브 호러
플라이 벤처
회전 목마

답

- [ch05/park.py](#)

5.2.5 연습 문제: 놀이 공원 (3)

문제

이 문제를 풀기 전에 [놀이 공원 \(1\)](#)과 [놀이 공원 \(2\)](#) 문제를 먼저 풀어야 합니다.

놀이 기구 중에서 둘리, 도우너, 마이콜이 모두 탈 수 있는 것을 출력합니다.

입출력 예

입력:

```
120 110 179
```

출력:

```
와일드윙  
툼오브호러  
회전목마
```

답

- [ch05/together.py](#)

5.3 여러 가지 모듈

이번 시간도 계속해서 모듈에 대해 알아볼까요? 파이썬에서 기본적으로 제공하는 수많은 모듈 중에서 자주 쓰이는 것들을 이번 시간에 살짝 소개해 드리려고 합니다.

sys

처음으로 알려드릴 것은 sys 모듈입니다. 요놈은 파이썬 인터프리터를 제어할 수 있는 방법을 제공하지요.

파이썬 인터프리터를 띄워주세요. 인터프리터가 우리의 명령을 기다린다는 뜻으로 >>>를 표시하고 있죠? 도스와 마찬가지로 이것도 프롬프트라고 합니다.

sys 모듈을 사용하면 이 프롬프트를 바꿀 수가 있지요.

```
>>> import sys
>>> sys.ps1                                     # 현재의 프롬프트는?
'>>> '
>>> sys.ps1 = '^_^; '
'^_^; print('hello')
hello
'^_^; 5 * 3
15
'^_^;
```

재미있지요? 이번엔 인터프리터를 끝내볼까요?

```
'^_^; sys.exit()
```

os

그 다음에는 os 모듈을 보겠습니다. 이것은 운영체제 (OS : Operating System)를 제어할 수가 있지요.

파이썬을 새로 실행하고, 현재 경로를 알아볼게요.

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.getcwd()
'C:\Users\Yong Choi\AppData\Local\Programs\Python\Python38-32'
```

os 모듈의 `getcwd()`는 현재 작업 디렉터리를 알려줍니다. 구해라 (get)! 무엇을? 현재 작업 디렉터리 (**current working directory**) 를 ~

이곳에 어떤 파일들이 있는지 알아볼까요?

이곳에 어떤 파일들이 있는지 알아볼까요?

```
>>> os.listdir()
['DLLs', 'Doc', 'img_read.py', 'include', 'Lib', 'libs', 'LICENSE.txt',
'NEWS.txt', 'python.exe', 'python3.dll', 'python38.dll', 'pythonw.exe',
'Scripts', 'tcl', 'Tools', 'vcruntime140.dll']
```

이중에서 `libs` 디렉터리로 이동해보겠습니다.

```
>>> os.chdir('libs')
>>> os.getcwd()
'C:\Users\Yong Choi\AppData\Local\Programs\Python\Python38-32\libs'
```

현재 작업 디렉터리가 바뀌었죠? 파일 목록도 보겠습니다.

```
>>> os.listdir()
['python3.lib', 'python38.lib', '_tkinter.lib']
```

상위 디렉터리는 ..으로 나타냅니다.

```
>>> os.chdir('..')
>>> os.getcwd()
'C:\Users\Yong Choi\AppData\Local\Programs\Python\Python38-32'
```

re

정규 표현식 (regular expression) 을 이용해 문자열을 다룰 수 있는 `re` 모듈도 있어요. 다음 예제에서 두 번 째 줄의 괄호 안에 쓴 것이 정규 표현식인데요, 마침표(.) 는 문자 아무거나 한 개를 뜻하고, 별표 (*) 는 0 개 이상의 문자를 뜻합니다. 그래서 현재 디렉토리에서 p 다음에 n 이 나오는 이름을 갖고 있는 파일들을 모두 찾아주게 되지요. 실행한 결과를 잘 보시면 이해가 되실거예요.

```
>>> import re, glob
>>> p = re.compile('.*p.*n.*')
>>> for i in glob.glob('*'):
...     m = p.match(i)
...     if m:
...         print(m.group())
...
pycon.ico
python.exe
pythonw.exe
w9xpopen.exe
```

이런 것들 외에 처음에 모듈에 대해 설명드릴 때 보여드린 math 나 tkinter 도 자주 쓰실 법하네요.

지금까지 몇 가지 예를 보여드렸는데 모듈들이 참 쓸만하죠? 파이썬에서 제공하는 모듈을 잘 활용하면 좋은 프로그램을 쉽게 만들 수 있을 것 같네요. 하지만 수 많은 모듈의 사용법을 모두 머리에 집어넣으실 필요는 없겠죠? 작성하실 프로그램에서 어떤 기능을 필요로 하는가에 따라 어떤 모듈을 사용할 것인지 결정한 다음, 사용설명서를 보면서 모듈의 사용법을 익혀서 프로그래밍하시면 됩니다. 모듈의 사용설명서로는 파이썬과 함께 기본적으로 설치되는 'Python Library Reference(파이썬 라이브러리 레퍼런스)'라는 것도 있고, 책이나 인터넷을 통해 자료를 찾아볼 수도 있지요.

webbrowser

끝으로 재미있는 모듈을 하나 더 소개해드릴게요.

한 번 따라해보세요. 그럼 전 이만... 휘리릭 ~

```
>>> import webbrowser
>>> url='http://www.python.org/'
>>> webbrowser.open(url)
True
>>>
```

5.3.1 랜덤 (random) 모듈

이번에는 파이썬에서의 랜덤 (random)에 대해 가볍게 정리해볼까 합니다.

우선 랜덤이 무엇인지부터 살펴볼까요.

주사위를 던지는 상황을 생각해봅시다. 주사위의 각 면에는 1개에서 6개까지의 눈이 새겨져 있어서, 주사위를 던질 때마다 그 중 하나의 숫자가 선택됩니다.

주사위를 직접 던져보기 전에는 다음번에 어떤 숫자가 나올지 알 수가 없죠.

그런데 주사위를 600 번 정도 던져보면 각 숫자가 대략 100 번 정도는 나오기는 합니다.

이런 것이 바로 난수 (random number)입니다.

난수의 예가 될 만한 것으로 주사위 외에 또 어떤 것들이 있을까요? 복권 추첨, 음악 재생 순서 섞기...

그럼 파이썬으로 난수를 만들어봅시다.

```
>>> import random  
>>> random.random()  
0.90389642027948769
```

random 모듈의 random() 함수를 호출했더니 복잡한 숫자를 돌려주네요. random() 함수는 0 이상 1 미만의 숫자 중에서 아무 숫자나 하나 뽑아서 돌려주는 일을 한답니다.

주사위처럼 1에서 6까지의 정수 중 하나를 무작위로 얻으려면 어떻게 해야 할까요? 이럴 때 편리하게 쓸 수 있는 randrange()라는 함수가 있습니다.

```
>>> random.randrange(1,7)  
6  
>>> random.randrange(1,7)  
2
```

여기에서 randrange(1,6)이 아니라 randrange(1,7)이라고 썼다는 점에 주의하세요.

“1 이상 7 미만의 난수”라고 생각하시면 이해가 쉽습니다.

내장함수인 **range()**를 되새겨보는 것도 좋겠군요.

```
>>> range(1,7)
[1, 2, 3, 4, 5, 6]
```

shuffle()이라는 재미있는 함수도 있군요. 시퀀스를 뒤죽박죽으로 섞어놓는 함수입니다.

```
>>> abc = ['a', 'b', 'c', 'd', 'e']
>>> random.shuffle(abc)
>>> abc
['a', 'd', 'e', 'b', 'c']
>>> random.shuffle(abc)
>>> abc
['e', 'd', 'a', 'c', 'b']
```

아무 원소나 하나 뽑아주는 **choice()** 함수도 있네요.

```
>>> abc
['e', 'd', 'a', 'c', 'b']
>>> random.choice(abc)
'a'
>>> random.choice(abc)
'd'

>>> menu = '쫄면', '육계장', '비빔밥'
>>> random.choice(menu)
'쫄면'
```

참과 거짓 중에 하나를 뽑고 싶다면, 뭐.. 까짓 거... 이렇게 해주죠...

```
>>> random.choice([True, False])
True
>>> random.choice([True, False])
False
```

5.3.2 연습 문제: 밴드 이름 짓기 (1)

문제

색깔 이름과 음식 이름을 더하면 밴드 이름 같다고 합니다.¹

색 이름과 음식 이름을 무작위로 뽑아서 밴드 이름을 지어주는 프로그램을 만들어 보세요.

예

```
$ python bandname1.py  
검은 과 쑥 침  
$ python bandname1.py  
화이트 소 보로  
$ python bandname1.py  
파란 쭈 꾸 미  
$ python bandname1.py  
청색 커 피  
$ python bandname1.py  
파란 옹 심 이
```

답

- 코드: [ch05/bandname1.py](#)

¹지금입고있는 하의 색과 마지막으로 먹은음식이름을 더하면 밴드이름같다

5.3.3 string 과 random 모듈을 이용해 비밀번호 생성

여러 인터넷 사이트와 앱을 이용하다보면 비밀번호 (password) 를 만들고 입력할 일이 많죠. 여러분은 비밀번호를 어떻게 관리하시나요?

비밀번호를 통일하면 관리하기 편하겠지만, 공격자가 한 곳의 비밀번호만 알아내면 다른 곳의 비밀번호도 모두 알게 되므로, 사이트마다 다른 패스워드를 쓰는 것이 좋습니다. 또, 사이트마다 비밀번호가 다르더라도 일정한 규칙에 따라 만들어져 있으면 공격자가 유추할 가능성이 있고요. 다행히 요즘에는 비밀번호를 만들고 기억해주는 기능이 웹 브라우저와 스마트폰에 있어서 비밀번호 관리가 한결 수월해졌습니다.

이번에는 파이썬으로 비밀번호를 만들어보려고 합니다.

string 모듈

먼저 재료를 준비하겠습니다. 일반적으로 비밀번호는 영문자와 숫자, 특수문자를 섞어서 만들죠.

영문자부터 해볼게요. “ABCD...” 를 하나하나 타자해도 되겠지만, `string` 모듈을 이용하겠습니다. `string` 모듈을 임포트해주세요.

```
>>> import string
```

`string.ascii_uppercase`에는 대문자 A부터 Z 까지가 있습니다.

```
>>> string.ascii_uppercase  
'ABCDEFGHIJKLMNPQRSTUVWXYZ'
```

마찬가지로 `string.ascii_lowercase`에는 영문 소문자가 있습니다.

```
>>> string.ascii_lowercase  
'abcdefghijklmnopqrstuvwxyz'
```

참, 소문자와 대문자를 모두 갖고 있는 `string.ascii_letters`도 있어요.

```
>>> string.ascii_letters  
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

이번에는 숫자를 준비합시다. `string.digits`를 쓰면 됩니다.

```
>>> string.digits  
'0123456789'
```

영문자와 숫자를 한데 모아 `alphanumeric` 변수를 만들겠습니다.

```
>>> alphanumeric = string.ascii_letters + string.digits  
>>> alphanumeric  
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
```

몇 글자인지 세어볼까요?

```
>>> len(alphanumeric)  
62
```

소문자 엘 (`l`) 과 대문자 아이 (`I`) 가 비슷하게 생겨서 헷갈리니까 그 두 개는 빼는 게 좋겠어요. 그리고 숫자 0과 대문자 오 (`O`) 도 헷갈릴 수 있으니 빼고요. 이럴 때 `세트`를 쓰면 좋겠네요.

```
>>> list(set(alphanumeric) - set('lIO0'))  
['L', 'k', 'g', 'j', 'T', '5', 'Z', 'a', 'M', 'w', 'D', 'P', 'n', 'f', 'x', 'd', '  
q', 'K', 'J', 'X', '4', 'b', 'F', 'Y', 'A', 'v', 'r', 'i', 'o', '7', 'y', 'S',  
'1', 'u', 'W', 'V', 'R', 'G', 'h', 'c', 'N', 'U', '2', '6', 'C', 'B', 'e', 'p',  
's', 'z', 'H', '9', 't', '8', 'E', 'm', '3', 'Q']
```

특수문자는 밑줄만 쓰는 걸로 할까요?

```
>>> chars = list(set(alphanumeric) - set('lIO0')) + ['_']
```

비밀번호를 만들 때 쓰고 싶은 글자 59 개로 이뤄진 `chars`라는 이름의 리스트가 준비됐습니다.

```
>>> len(chars)  
59
```

문자열 섞기

재료를 준비했으니 비밀번호를 만들어 보겠습니다. 글자를 무작위로 고르려면 앞에서 배운 [random 모듈](#)을 쓰면 되겠죠?

```
>>> import random
```

아무 글자나 하나 뽑아볼게요.

```
>>> random.choice(chars)  
'Y'
```

또 뽑아볼게요.

```
>>> random.choice(chars)  
'Q'  
>>> random.choice(chars)  
'V'  
>>> random.choice(chars)  
'x'
```

이런 식으로 반복해서 비밀번호를 만들면 되겠군요.

```
>>> pw = str()  
>>> for i in range(16):  
...     pw += random.choice(chars)  
...  
>>> pw  
'yTjFDnEaLvE8S2uo'
```

16 자로 된 비밀번호를 만들어봤습니다.

그런데 밑줄은 들어있지 않네요. 그도 그럴 것이, 무작위로 뽑으면 각 글자가 1/59 확률로 나올 테니 밑줄이 나오기 힘들 것 같네요.

영문자, 숫자, 특수문자를 모두 넣어야 한다는 규칙이 있다면 좀 더 머리를 써야 할 것 같습니다. 똑똑한 여러분이 한번 도전해보세요. 저는 곧 치킨 배달이 와서 이만....

P.S. 저는 이렇게 만들어봤습니다. [ch05/gen_password.py](#)

5.3.4 시저 (카이사르) 암호 만들기

로마의 율리우스 카이사르가 사용했다고 해서 ‘카이사르 암호’ 또는 ‘시저 암호’로 알려진 것을 파이썬으로 만들어보겠습니다.

시저 암호의 원리

카이사르 암호에서는 알파벳의 각 글자를 다른 글자로 바꾸는 표를 만듭니다. 다음 표의 위 줄은 소문자를 a부터 알파벳 순으로 늘어놓은 것이고, 아래 줄은 대문자 중 네 번째 글자인 D부터 시작해서 알파벳 순서로 늘어놓은 것입니다. 아래 줄의 Z 다음에는 다시 A가 옵니다.

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
| D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C |

암호화하고 싶은 문장의 각 글자에 대해, 표의 위 줄에 있는 글자를 아래 줄에 있는 글자로 바꾸면 암호문이 만들어집니다. 편의상 평문은 소문자로, 암호문은 대문자로 표시하겠습니다.

평문 ('절대로 브루투스를 신뢰하지 마라' 라는 뜻):

traue nie dem brutus

암호문:

WUDXH QLH GHP EUWXV

메시지를 보내는 사람과 받는 사람만 이 규칙을 알고 있으면, 전령이 메시지를 전달하는 도중에 적이 가로채더라도 뜻을 알기 어렵겠죠? 똑똑한 여러분이라면 금세 알아챌지도 모르겠네요.

암호표는 다음과 같이 두 개의 원판으로 나타낼 수도 있습니다. 안쪽 원판을 돌려서 암호 규칙을 바꿀 수 있겠죠?



파이썬으로 시저 암호 구현

먼저 암호표의 위 줄 (바깥쪽 원판)을 만들기 위해 a부터 z 까지의 영문 소문자 알파벳을 차례대로 늘어놓습니다. 직접 타자해도 되지만 `string` 모듈을 이용하겠습니다.

```
>>> import string  
>>> string.ascii_lowercase  
'abcdefghijklmnopqrstuvwxyz'
```

이번에는 암호표의 아래 줄 (안쪽 원판)에 해당하는 영문 대문자 D~Z, A~C 입니다.

```
>>> string.ascii_uppercase[3:] + string.ascii_uppercase[:3]  
'DEFGHIJKLMNOPQRSTUVWXYZABC'
```

위 줄과 아래 줄에 해당하는 문자열들을 준비했으니, 각 글자를 대응시키는 암호표에 해당하는 것을 만들어보겠습니다. 반복문을 써서 직접 만드는 방법도 있지만, 파이썬에서는 `str` 객체의 `maketrans()` 메서드를 이용해 쉽고 빠르게 구현할 수 있습니다. ('메서드'에 관해서는 7장에서 다룹니다. 지금은 그냥 함수와 비슷한 것이라고 생각하면 됩니다.)

```
>>> tt = str.maketrans(string.ascii_lowercase, string.ascii_uppercase[3:] + string.ascii_uppercase[:3])
```

`str.maketrans()`의 결과로 반환되는 딕셔너리에 `tt`라는 이름을 붙였습니다.

이제, 암호화하고 싶은 문자열을 `translate()` 메서드를 써서 암호화합니다. 이때 앞에서 만든 `tt`를 인자로 줍니다.

```
>>> 'traue nie dem brutus'.translate(tt)
'WUDXH QLH GHP EUWXV'
```

참 쉽죠?!

참고

- 루돌프 키펜한 지음 / 이일우 옮김, 《암호의 해석》

5.3.5. 연습 문제: 끝말 잇기 (2)

끝말 잇기를 하는 프로그램을 작성하세요. 규칙은 끝말 잇기 (1)과 같되, 두음 법칙을 적용합니다.

예

예 1~3 은 끝말 잇기 (1) 에서와 같고, 예 4 는 다릅니다.

예 1

| | |
|------|-------------------------|
| 턴 | 입출력 |
| 컴퓨터 | 끝말잇기를 하자. 내가 먼저 말할게. 기차 |
| 플레이어 | 자동차 |
| 컴퓨터 | 글자가 안 이어져. 내가 이겼다! |

예 2

| | |
|------|-------------------------|
| 턴 | 입출력 |
| 컴퓨터 | 끝말잇기를 하자. 내가 먼저 말할게. 기차 |
| 플레이어 | 차박 |
| 컴퓨터 | 박쥐 |
| 플레이어 | 쥐구멍 |

턴 입출력

컴퓨터 명계

플레이어 게시판

컴퓨터 모르겠다. 내가 졌어.

예 3

턴 입출력

컴퓨터 끝말잇기를 하자. 내가 먼저 말할게. 기차

플레이어 차주

컴퓨터 주기

플레이어 기차

컴퓨터 아까 했던 말이야. 내가 이겼어!

예 4

턴 입출력

컴퓨터 끝말잇기를 하자. 내가 먼저 말할게. 기차

플레이어 차량

컴퓨터 양심

플레이어 심술

컴퓨터 모르겠다. 내가 졌어.

풀이

- 코드: [ch05/wordgame2.py](#)

5.3.6 연습 문제: 내일의 날짜 구하기 (2)

문제

내일의 날짜 구하기 (1) 문제를 [datetime 모듈](#)을 활용해서 풀어 보세요.

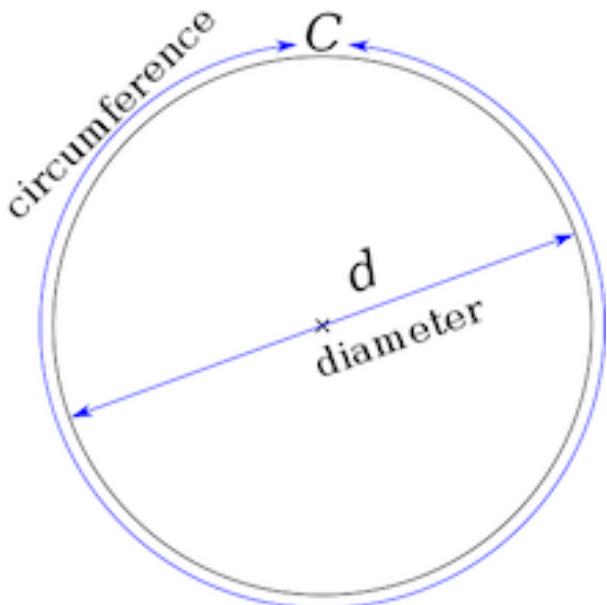
- [strftime\(\)](#) 과 [strptime](#)
- [timedelta](#)

답

- 코드: [ch05/tomorrow2.py](#)

5.3.7 연습 문제: 원주율 구하기

원 (circle)의 지름 (diameter)에 대한 둘레 (circumference)의 비를 ‘원주율’이라고 하죠.



원이 크든 작든, 모든 원에 대해 원주율은 약 3.14로 일정합니다. 그래서 수학에서 π [파이]라는 상수로 표시합니다.

앞에서도 봤듯이, 파이썬의 math 모듈에는 원주율 값이 들어 있어서 우리가 힘들게 계산할 필요가 없어요.

```
>>> import math  
>>> math.pi  
3.141592653589793
```

굳이 하지 않아도 되지만, 직접 구해보는 것도 재미있겠네요.

원주율을 구하는 원리

원주율을 계산하는 방법은 여러 가지가 있다고 하네요. 여기서는 무작위로 점을 찍어서 개수를 세는 방식으로 해보려고 합니다.

거북이로 원과 정사각형 그리기

잠시 후에 수학 공식을 설명하려고 하는데, 그림이 있으면 이해가 잘될 듯해서 거북이 (터틀 그래픽스)의 도움을 받아 그림을 그려 보겠습니다.

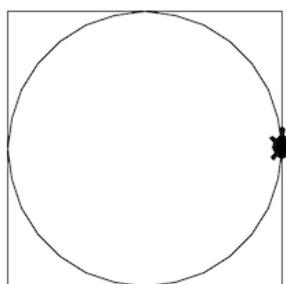
```
from turtle import *
shape('turtle')

# 원의 반지름
r = 100

# r만큼 이동해서 그리기 준비
up(); forward(r); down(); left(90)

# 반지름이 r인 원 그리기
circle(r)

# 한 변의 길이가 r의 두 배인 정사각형 그리기
# 한 번에 2r씩 직진해도 되지만 아래처럼 해도 되겠죠?
forward(r); left(90); forward(r); # 오른쪽 아래
forward(r); left(90); forward(r); # 오른쪽 위
forward(r); left(90); forward(r); # 왼쪽 위
forward(r); left(90); forward(r); # 왼쪽 아래
```



거북이에게 여러 가지 그림을 그려달라고 시키는 방법을 아래 동영상에 소개했으니 궁금하신 분은 참고하세요.

- <https://youtu.be/ZEV3pGTdlxw>

몬테카를로 방법으로 원주율을 구하는 원리

다시 원주율 구하는 법 이야기로 돌아갈게요.

그 이름 때문에 왠지 어려운 느낌이 들지만, ‘몬테카를로’는 그냥 동네 이름입니다. 우리나라로 치면 강원도 정선과 비슷한 곳입니다.

몬테카를로 방법으로 원주율을 구하는 원리를 이해하려면, 먼저 두 가지 수학 공식을 알아야 합니다.

1. 정사각형의 넓이는 한 변의 길이를 제곱한 값과 같다. ($= r^2$)
2. 원 넓이는 반지름 r 을 제곱한 값에 원주율을 곱한 것과 같다. ($= \pi r^2$)

정사각형의 한 변 길이가 원의 지름 (반지름의 두 배, 즉 $2r$) 과 같다면, 위의 첫 번째 공식을 다음과 같이 나타낼 수 있겠죠?

$$= (2r)^2 = 4r^2$$

그래서 원 넓이를 정사각형 넓이로 나눠보면 다음과 같이 됩니다.

$$\frac{\text{---}}{4} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

그래서 원주율은

$$\pi = 4\text{---}$$

이 됩니다.

그렇다면 원 넓이와 정사각형 넓이를 알면 원주율을 알 수 있겠군요!

정사각형 넓이는 구하기 쉬운데, 원 넓이를 어떻게 구하죠?

반지름 제곱에 원주율을 곱하 ♡ 아, 이게 아니지 ♡. -_-; ㅋ

이때 ‘까짓 거 대충 찍자!’ 라는 방법이 바로 몬테카를로 방법입니다.

어떻게 하냐면요, 정사각형 안의 아무 데나 점을 마구 찍은 다음에, 그 점이 원에 들어가는지 아닌지를 따져보는 거예요. 그래서 원 안에 있는 점의 개수가 전체에서 얼마나 되는지 비율을 구하면, 정확하지는 않아도 비슷하게나마 원주율을 구할 수 있다는 겁니다.

$$\approx 4 \frac{\text{원 안에 들어간 점의 개수}}{\text{한 변의 길이가 원의 지름과 같은 정사각형 안의 점의 개수}}$$

정말로 원주율의 근삿값을 구할 수 있는지는, 한번 코딩해서 확인해 보도록 하죠.

random 모듈을 이용해 원주율 구하기

사각형 안에 점을 찍는 방법부터 알아봐야 겠네요.

사각형 안에 점 찍기

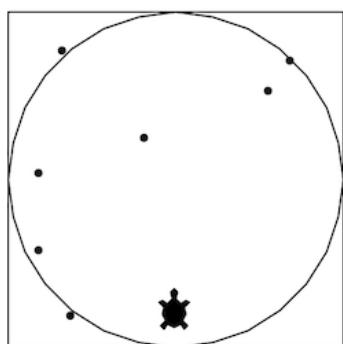
random 모듈을 이용해서, 거북이에게 사각형 안의 아무 곳에나 가서 점을 찍으라고 시켜 보겠습니다.

```
# 자취를 남기지 않기
penup()

from random import randrange

def draw_random_dot():
    point = randrange(-r, r), randrange(-r, r)
    goto(point)
    dot()

# 점 찍기
draw_random_dot()
draw_random_dot()
draw_random_dot()
draw_random_dot()
...
```



점이 몇 개인지 직접 세어 볼까요? 점이 총 8 개고 (한 개는 거북이에 가려짐) 그중에 6 개가 원 안에 있네요.

앞에서 만든 원주율 구하는 식에 넣어 보면 $4 \times \frac{6}{8}$ 이 되죠.

파이썬으로 계산하면,

```
>>> 4 * (6 / 8)
3.0
```

원주율은 약 3 이 됩니다!

점이 원 안에 있는지 판별하기

점을 더 많이 찍어서 실험하면 원주율을 더 정확하게 구할 수 있을 것 같은데, 점을 세기가 힘들어 질 것 같군요. 그래서 원 안에 있는 점의 개수를 세는 일을 파이썬에게 시켜보려고 합니다.

원 안에 들어간 점이 몇 개인지 알려면, 점이 원 안에 있는지 원 밖에 있는지를 구분할 수 있어야겠죠. 그래서 수학 지식이 또 필요합니다.

‘원 중심과 점 사이의 거리가 원의 반지름보다 작거나 같으면 점이 원 안에 있다’ 라고 정하기로 하죠.

원 중심이 $(0, 0)$ 에 있고 점 P 가 (a, b) 에 있다고 하면, 원 중심과 점 P 사이의 거리 d 는 다음과 같이 구할 수 있어요.

$$d = \sqrt{a^2 + b^2}$$

이것을 이용해서, 점의 x, y 좌표와 반지름 r 을 입력하면 그 점이 원의 안에 있는지 아닌지를 알려주는 함수를 만들어 볼게요.

```
import math

def in_circle(x, y, r):
    return math.sqrt(x ** 2 + y ** 2) <= r
```

그리고 앞에서 만든 점 찍기 함수를 고쳐보겠습니다.

```
def draw_random_dot():
    x, y = randrange(-r, r), randrange(-r, r)
    goto(x, y)
    if in_circle(x, y, r):
        color('red') # 원 안에 있으면 빨간색 점
    else:
        color('blue') # 원 밖에 있으면 파란색 점
    dot()
```

그림을 지우고 처음부터 다시 그려 볼게요.

```
# 재 설정
reset()

# r만큼 이동해서 그리기 준비
up(); forward(r); down(); left(90)

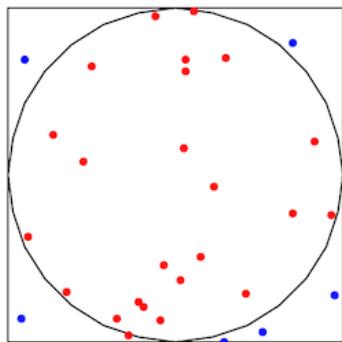
# 반지름이 r인 원 그리기
circle(r)

# 한 변의 길이가 r의 두 배인 정사각형 그리기
for i in range(4):
    forward(r); left(90); forward(r)

# 자취를 남기지 않기
penup()

# 점 30개 찍기
for i in range(30):
    draw_random_dot()

# 거북이 숨기기
hideturtle()
```



문제

점이 원 안에 있는지 검사하는 건 해결했으니, 이제 변수를 만들어서 점이 원 안에 있으면 변수값을 증가시키라고 하면 되겠네요. 그 부분은 여러분이 직접 코딩해서 완성해 주세요.

풀이

- [ch05/montecarlo.py](#)

참고

- <https://ko.wikipedia.org/wiki/%EA%B0%80%EB%A0%88>
- <https://en.wikipedia.org/wiki/Pi>
- [Estimating \$\pi\$ using the Monte Carlo Method](#)

몬테카를로 방법의 유래

몬테카를로 (Monte-Carlo) 는 도시국가인 모나코 북부에 있는 지역으로서 카지노, 도박으로 유명한 곳이기도 하다. 수학에서 확률이론의 탄생이 원래 도박에서 비롯되었는데, 과거 도박 도시의 대명사였던 몬테카를로 역시 지금은 확률론과 밀접한 관계가 있는 과학적 방법론을 지칭하기도 한다.

오늘날 다양한 과학기술 분야의 컴퓨터 시뮬레이션, 예측 등에 빈번히 쓰이는 몬테카를로 방법이 적용된 매우 유명한 역사적 사례로서, 미국의 원자폭탄 개발 계획인 맨해튼 프로젝트를 들 수 있다. 바로 몬테카를로 방법이라는 이름이 처음 쓰이게 된 계기이기도 하다.

즉 폴란드 출신의 수학자 스탠리슬로 울람 (Stanisław Marcin Ulam, 1909-1984) 은 컴퓨터의 아버지로 잘 알려진 폰 노이만 (Johann Ludwig von Neumann, 1903-1957) 등과 함께 맨해튼 프로젝트에 참여하였다.

예전부터 노이만의 동료였던 울람은 극비의 코드명에 적합하도록 새로운 수학적 방법론을 도박의 도시 이름을 따서 몬테카를로 방법이라 명명하였고, 이 방법은 중성자가 원자핵과 충돌하는 과정을 이해하고 묘사하는 데에 결정적 역할을 하였다.

- 최성우, [『몬테카를로 방법과 인공지능』](#), 사이언스타임즈

6. 파일

파이썬으로 파일을 다루는 법을 알아봅니다.

이 장에서 배우는 것:

- 텍스트 파일
- 한 줄씩 다루기
- 파일을 입맛대로 (pickle, glob, os.path)
- 응용 예제: 음성 인식을 활용한 일본어 퀴즈



6.1 텍스트 파일

이번 시간에는 파일을 다루는 방법을 알아보겠습니다. 파일명을 바꾼다거나, 파일을 복사하고, 지우는 일들은 지난 시간에 살펴본 os 모듈이나, shutil이라는 모듈을 사용하면 되죠. 지금 배울 것은 그런 것이 아니라 파일의 내용을 읽고, 쓰는 방법입니다.

파일을 다룰 수 있게 되면 프로그램과 데이터를 따로 관리할 수 있지요. 만약 주소록 프로그램을 만든다면 연락처를 담은 데이터 파일을 따로 만들어두었다가, 새 친구가 생길 때마다 그 데이터 파일에 연락처를 추가해주면 되겠죠?

먼저 메모장으로 텍스트 파일 하나를 만들어봅시다.

```
Programming is fun.  
Very fun!  
  
You have to do it yourself...
```

각자 파이썬 스크립트를 저장하기 위한 폴더를 만들어두셨을 거예요. 거기에도 Python_for_Fun.txt 와 같은 이름으로 저장해주세요. 그리고나서 아래 예제를 따라해보세요. 폴더와 파일의 이름은 각자 지으신 대로 써주시구요.

```
>>> f = open('C:\\python_newbie\\Python_for_Fun.txt')  
>>> f.read()  
'Programming is fun.\nVery fun!\n\nYou have to do it yourself.'
```

파일을 열어서 f라는 이름을 붙여주고 다시 그것을 읽었죠? 여러분도 잘 읽어지나요?

그런데, 좀 이상한 것이 있죠? 아까 파일을 작성할 때 줄을 바꿔가면서 썼던 것이 모두 한 줄에 나옵니다. 쓴 적이 없는 글자도 끼어 있구요.

잘 보면 줄을 바꾼 곳마다 '\\n'이 들어있다는 것을 알 수 있지요. '\\n'은 다음과 같이 문자열을 출력할 때 '줄 바꿈'을 의미한답니다.

```
>>> print('야 호~\\n호 야~')
```

야 호~
호 야~

우리가 메모장에서 텍스트 파일을 작성할 때에도 줄을 바꿀 때마다 이런 개행 문자가 포함되었던 것이지요. 위에서는 f.read() 함수가 돌려준 문자열을 그대로 보았기 때문에 우리가 개행 문자도 볼 수 있었던 것이고, 이 문자열을 print 하면 정상적으로 줄 넘김이 된 상태로 보입니다.

```
>>> buffer = f.read()
>>> print(buffer)
Programming is fun.
Very fun!

You have to do it yourself...
```

이번엔 파일에 글을 써볼까요?

```
>>> letter = open('C:\\python_newbie\\letter.txt', 'w') # 새 파일 열기
>>> letter.write('Dear Father,')                      # 아버님 전 상서
>>> letter.close()                                     # 닫기
```

이번엔 파일명 뒤에 'w' 라고 써주었죠? 아까 파일의 데이터를 읽기만 했을 때와는 달리, 파일에 데이터를 쓰려고 할 때는 미리 '파일에 데이터를 쓰겠다' 는 사실을 알려줄 필요가 있기 때문입니다.

그 다음에 몇 자 적어주고 나서 파일을 닫았습니다.

이제 해당 폴더를 보시면 letter.txt 파일이 있구요, 그걸 메모장으로 열어보시면 파일에 데이터가 고스란히 적혀있을 거예요. 훌륭하죠?

만약, 파일에 데이터를 쓴 다음에 close() 로 닫아주지 않았으면 메모장으로 봐도 아무 글자도 없을 거예요. 그러니 꼬옥 ~ 닫아주셔야 해요 ~.

예리하신 분들은 그 전의 예제에서 파일을 읽었을 때는 왜 안 닫았는지 궁금해 하시겠죠? 닫아주면 좋답니다.^^ 하지만 우리가 잊어버리더라도 파일이 더 이상 필요 없으면 파일이 알아서 닫아준다고 하네요. 참 편하죠?

혹시 letter.txt 가 아니라 python.txt 파일에다가 'Dear Father' 라고 쓰신 분 계신가요? 분명히 계실 것 같은데...

원래 들어있던 내용이 다 날아갔죠? 시키는대로 안 하니까 그렇죠. 히히히

하지만, 전 시키는 대로만 하는 사람은 싫어합니다. 잘 하셨어요.

파일에 데이터가 원래 들어있을 때 ‘w’ 모드로 파일을 열면 원래 있던 데이터가 없어져버린답니다. 파일에 들어있는 데이터를 지우지 않고 내용을 추가하려면 ‘a+’ 모드를 지정해줘야 하죠.

```
>>> letter = open('C:\\python_newbie\\letter.txt', 'a+')
>>> letter.write('\\n\\nHow are you?')
>>> letter.close()
```

이제 letter 를 다시 열어보시면 제대로 되어있을 거예요.

오늘 배운 것은 간단하면서도 아주 쓸모 있답니다. 이것을 가지고 무엇을 할 수 있을지 한번 생각해보세요.

6.2 한 줄씩 다루기

지난 시간에 이어 오늘도 텍스트 파일을 괴롭혀 보겠습니다. 오늘은 한 줄씩 난도질을... -+

파이썬이 설치된 디렉터리에 있는 `LICENSE.txt` 파일을 열어보겠습니다.

```
>>> import os
>>> os.getcwd()
'C:\\\\Users\\\\Yong Choi\\\\AppData\\\\Local\\\\Programs\\\\Python\\\\Python38-32'
>>> os.listdir()
['DLLs', 'Doc', 'img_read.py', 'include', 'Lib', 'libs', 'LICENSE.txt', 'NEWS.txt',
 ,
 'python.exe', 'python3.dll', 'python38.dll', 'pythonw.exe', 'Scripts', 'tcl',
 'Tools', 'vcruntime140.dll']
>>> f = open('LICENSE.txt')
```

파일을 처음부터 끝까지 읽을 땐 `read()`를 썼죠? 한 줄씩 읽을 때는 `readline()`을 사용하면 됩니다.

첫 줄을 읽어볼게요.

```
>>> f.readline()
'A. HISTORY OF THE SOFTWARE\\n'
```

다음 줄도 읽어볼까요?

```
>>> f.readline()
'=====\\n'
```

간단하죠? 별 것 아닙니다. 파일 내용을 처음부터 한 글자씩 주루룩 ~ 읽어나가다가 '`\\n`'이 나타나면 한 줄이 끝난 줄 알고 딱 멈춰서게 되는 겁니다.

다음과 같이 반복문과 `readline()`을 함께 사용해 텍스트를 원하는 줄 수 만큼만 읽어들일 수 있답니다.

```
>>> for x in range(5):
```

```
...     f.readline()
...
'\n'
'Python was created in the early 1990s by Guido van Rossum at Stichting\n'
'Mathematisch Centrum (CWI, see http://www.cwi.nl) in the Netherlands\n'
"as a successor of a language called ABC. Guido remains Python's\n"
'principal author, although it includes many contributions from others.\n'
```

파일을 한 줄씩 읽어 화면에 출력하기를 다섯 번 되풀이했지요.

이번엔 똑같이 다섯 줄을 읽기는 하지만 조금 다르게 하는 방법을 보여드리겠습니다.

```
>>> f = open('LICENSE.txt')
>>> lines = f.readlines()
>>> lines[:2]
['A. HISTORY OF THE SOFTWARE\n', '=====\\n']
```

여기서는 ‘readline’에 ‘s’ 자가 붙은 `readlines()`를 사용했습니다. `readlines()`로 파일을 읽으면 한 줄, 한 줄이 각각 리스트의 원소로 들어갑니다.

위에서는 파일 전체가 `lines`라는 리스트에 쭉 담겼습니다. 그런 다음엔 `lines`에 들어있는 것들을 입맛대로 꺼낼 수 있습니다.

이 방법을 쓰면 아주 쉽게 원하는 줄을 읽어들일 수 있겠죠? 26 번째 줄부터 40 번째 줄까지를 읽어볼까요?

```
>>> for l in lines[26:41]:
...     print(l, end=' ')
...
   Release      Derived      Year      Owner      GPL-
   from
                                         compatible? (1)
   0.9.0 thru 1.2          1991-1995    CWI      yes
   1.3 thru 1.5.2    1.2          1995-1999    CNRI     yes
   1.6                  1.5.2        2000      CNRI      no
   2.0                  1.6          2000  BeOpen.com  no
   1.6.1                1.6          2001      CNRI  yes (2)
   2.1                  2.0+1.6.1    2001      PSF      no
   2.0.1                2.0+1.6.1    2001      PSF     yes
   2.1.1                2.1+2.0.1    2001      PSF     yes
   2.1.2                2.1.1        2002      PSF     yes
   2.1.3                2.1.2        2002      PSF     yes
   2.2 and above      2.1.1    2001-now    PSF     yes
>>>
```

왜 26:40이 아니라 26:41이라고 하는지 아리송하신 분은 [문자열과 리스트 강좌](#)를 복습하셔야겠네요.

오늘의 하이라이트! 끝에서 열 줄을 읽어봅시다.

대체 어떻게 해야 할까요? 파이썬 튜토리얼을 살펴보시면 직접 알아내실 수도 있는데...

힌트라도 드릴까요?

리스트에서 맨 마지막 원소의 인덱스는 -1이랍니다. 그게 무슨 말이징...

한번 도전해보세요. 성공하신 분은 다음과 같은 결과를 볼 수 있을 거예요 ~

Parts of this software are based on the Tcl/Tk software copyrighted by the Regents of the University of California, Sun Microsystems, Inc., and other parties. The original license terms of the Tcl/Tk software distribution is included in the [file](#) docs/license.tcltk.

Parts of this software are based on the HTML Library software copyrighted by Sun Microsystems, Inc. The original license terms of the HTML Library software distribution is included in the [file](#) docs/license.html_lib.

6.2.1 연습 문제: 밴드 이름 짓기 (2)

문제

색 이름과 음식 이름을 담은 텍스트 파일을 읽어서 밴드 이름을 지어 주는 프로그램을 만들어 보세요.

텍스트 파일을 만들려면 아래 링크를 오른쪽 클릭하고 [다른 이름으로 링크 저장] 을 선택합니다. 텍스트 파일과 파이썬 파일을 같은 폴더에 두세요.

- 색 이름¹: color.txt (<https://raw.githubusercontent.com/ychoi-kr/wikidocs-chobo-python/master/ch06/color.txt>)
- 음식 이름²: food.txt (<https://raw.githubusercontent.com/ychoi-kr/wikidocs-chobo-python/master/ch06/food.txt>)

예

```
$ python bandname2.py  
아이 보리 떡국  
$ python bandname2.py  
미드나이트블루 빙수  
$ python bandname2.py  
미디엄 터콰이즈 각두기  
$ python bandname2.py  
플럼 머스터드  
$ python bandname2.py  
라이트 그레이 디저트  
$ python bandname2.py  
씨쉘 계란덮밥
```

¹<https://supervitamin.tistory.com/76>

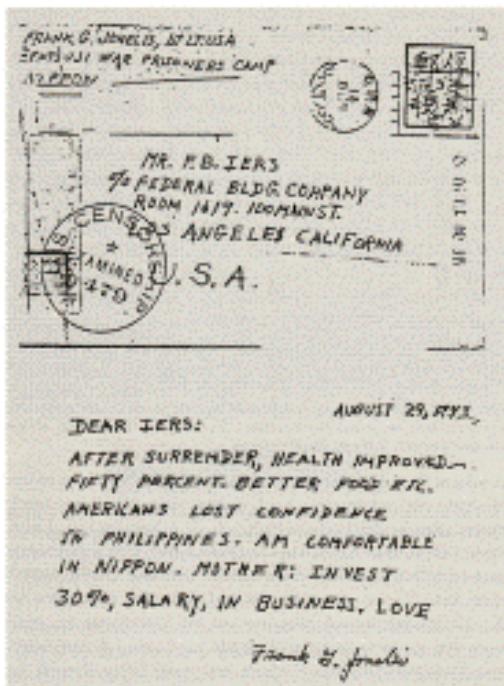
²https://ko.wiktionary.org/wiki/%EB%85%A5%EB%85%A5_%EB%A1%A4

답

- 코드: [ch06/bandname2.py](#)

6.2.2 연습 문제: 비밀 메시지

제 2 차 세계대전이 한창이던 1943년 미국 로스앤젤레스의 집배원은 엽서 한장을 집어들었습니다. 보낸 이는 일본의 전쟁포로 수용소에 수감돼 있던 미군 프랭크 조넬리스, 받는이는 페더럴 빌딩 회사 1619호의 F.B.I.ers 씨로 적혀 있었습니다. 회사도 사람도 실제로 존재하지 않았지만, 해당 주소의 619호에는 연방수사국(FBI) 사무실이 있었습니다.



(그림 출처: <http://informatik.rostfrank.de/info/lex09/lex0903.html>)

엽서의 내용은 다음과 같습니다.

August 29, 1943

Dear Iers:

After surrender, health improved
fifty percent. Better food etc.
Americans lost confidence
in Philippines. Am comfortable

```
in Nippon. Mother: Invest  
30%, salary, in business. Love  
  
(signed)  
Frank G. Jonelis
```

“항복 후, 건강이 50% 나아졌다. 좋은 음식 등. 미국인은 필리핀에서 자신감을 잃었다. 일본에서 편안하다. 어머니: 30% 투자, 봉급, 사업에. 사랑해”

문장이 약간 어색하긴 했지만 별다른 내용이 없어 보였습니다. 그렇지만 연방수사국에서는 여기에 다른 메시지가 숨겨져 있을지 모른다고 생각해 여러 가지 방법으로 단어를 배열해보다가, 각 줄의 첫 두 단어를 문장 부호 없이 조합하면 다음의 문장을 만들 수 있다는 것을 알아냈습니다.(마지막 단어 생략)

```
AFTER SURRENDER FIFTY PERCENT AMERICANS LOST IN PHILIPPINES IN NIPPON 30%
```

“항복한 후 필리핀에서 50% 일본에서 30% 의 미군을 잃었다”

문제

텍스트 편집기를 사용해 위의 엽서 내용을 `postcard.txt` 파일에 저장합니다.

1. 파일 읽기

`postcard.txt` 파일을 읽어 화면에 출력하세요.

2. 본문 추려내기

엽서 내용은 머리말, 본문, 꼬리말 형태로 되어 있고 그 사이는 한 줄씩 띄어져 있습니다. 본문 내용만 화면에 출력하세요.

출력

```
After surrender, health improved  
fifty percent. Better food etc.  
Americans lost confidence  
in Philippines. Am comfortable  
in Nippon. Mother: Invest
```

```
30%, salary, in business. Love
```

3. 문장부호 제거

본문에서 마침표 (.), 쉼표 (,), 콜론 (:) 을 제거해 출력하세요.

출력

```
After surrender health improved
fifty percent Better food etc
Americans lost confidence
in Philippines Am comfortable
in Nippon Mother Invest
30% salary in business Love
```

4. 대문자로 변환

3 번의 출력 결과를 모두 대문자로 나타내세요.

출력

```
AFTER SURRENDER HEALTH IMPROVED
FIFTY PERCENT BETTER FOOD ETC
AMERICANS LOST CONFIDENCE
IN PHILIPPINES AM COMFORTABLE
IN NIPPON MOTHER INVEST
30% SALARY IN BUSINESS LOVE
```

5. 비밀 메시지 출력

4 번의 결과에서 각 행의 처음 두 단어만 추려서 출력하세요.

```
AFTER SURRENDER FIFTY PERCENT AMERICANS LOST IN PHILIPPINES IN NIPPON 30% SALARY
```

답

- 엽서 내용: [ch06/postcard.txt](#)
- 파일 썬 코드: [ch06/secret_message.py](#)

참고

- 루돌프 키펜한 지음 / 이일우 옮김, 『암호의 해석』

6.2.2. 연습 문제: 끝말 잇기 (3)

끝말 잇기를 하는 프로그램을 작성하세요. 컴퓨터와 사용자는 텍스트 파일에 있는 단어를 사용해야 합니다.

다음 주소의 파일을 다운로드해서 사용하세요.¹

- https://raw.githubusercontent.com/ychoi-kr/wikidocs-chobo-python/master/ch06/korean_words.txt

그 밖의 규칙은 [끝말 잇기 \(2\)](#)와 같습니다.

풀이

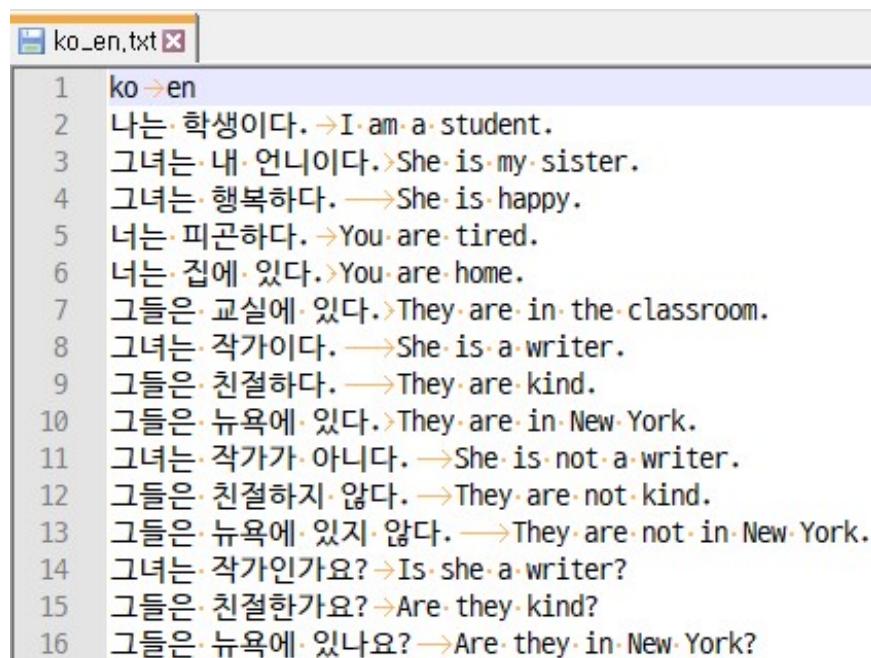
- 코드: [ch06/wordgame3.py](#)

¹ [Frequency lists by Neri](#) 블로그의 [The 2000 Most Frequently Used Korean Nouns](#)를 구글 스프레드시트로 가공한 뒤 수작업으로 손질하고 단어도 좀 더 추가했습니다.

6.2.3 연습 문제: 영어 퀴즈

문제

ko_en.txt 파일에는 우리말과 영어 문장이 탭으로 구분되어 있습니다.(예문 출처: <https://youtu.be/5AGUHzYzgw>)



ko_en.txt

```
1 ko → en
2 나는·학생이다. → I·am·a·student.
3 그녀는·내·언니이다. > She·is·my·sister.
4 그녀는·행복하다. → She·is·happy.
5 너는·피곤하다. → You·are·tired.
6 너는·집에·있다. > You·are·home.
7 그들은·교실에·있다. > They·are·in·the·classroom.
8 그녀는·작가이다. → She·is·a·writer.
9 그들은·친절하다. → They·are·kind.
10 그들은·뉴욕에·있다. > They·are·in·New·York.
11 그녀는·작가가·아니다. → She·is·not·a·writer.
12 그들은·친절하지·않다. → They·are·not·kind.
13 그들은·뉴욕에·있지·않다. → They·are·not·in·New·York.
14 그녀는·작가인가요? → Is·she·a·writer?
15 그들은·친절한가요? → Are·they·kind?
16 그들은·뉴욕에·있나요? → Are·they·in·New·York?
```

이 파일을 이용해 영어 퀴즈를 내는 프로그램을 작성하세요. 화면에 우리말로 된 문장을 출력한 다음, 사용자에게 영어 문장을 입력받고, 사용자가 답을 맞혔는지 화면에 표시합니다.

예:

```
Write the following sentence in English.  
그녀는 작가가 아니다.
```

```
your answer: She is not a writer.
```

```
result: Correct!
```

Write the following sentence **in** English.
그들은 교실에 있다.

your answer: They are **in** the classroom.

result: Correct!

Write the following sentence **in** English.
너는 집에 있다.

your answer: I am home.

result: Not correct!

right answer: You are home.

답

- 코드: ch06/english_quiz.py

6.3 파일을 입맛대로 (**pickle**, **glob**, **os.path**)

파일을 입맛대로 요리할 수 있도록 여러 가지 비법을 전수해드리지요.

```
## pickle
```

먼저 조금 복잡한 자료를 파일에 쓰고 읽는 방법부터 알아봅시다. 이럴 때는 **pickle**이란 모듈을 사용합니다. 피자 먹을 때 나오는 피클과 이름이 같네요.^^

예제로는 회원의 ID 와 비밀번호를 파일에 저장하는 것을 생각해보았습니다. 실제로 서비스를 만든다면 암호화해서 저장해야겠지만, 여기서는 평문 (plain text) 으로 저장할게요.

```
>>> users = {'kim':'3kid9', 'sun80':'393948', 'ljm':'py90390'}
>>> f = open('users', 'wb')
>>> import pickle
>>> pickle.dump(users, f)
>>> f.close()
```

처음에 ID 와 비밀번호를 **users**라는 딕셔너리에 담았습니다.

그리고 **users**라는 파일을 새로 열어서 **f**라는 이름을 붙였구요. 이때 **open()** 함수의 두 번째 인자인 **wb**는 일반 텍스트가 아닌 바이트 (**byte**) 형식으로 쓰겠다 (**write**) 는 뜻입니다.

그 다음에는 **pickle** 모듈의 **dump()**를 사용했습니다. 공사장에 흙을 실어나르는 덤프 트럭이 뒤쪽 짐칸을 들어올려서 흙을 와르르 쏟아내는 것처럼, **dump**는 **users**라는 리스트를 파일 **f**에 기록합니다.

파이썬이 실행되는 경로에 **users** 파일이 만들어진 것을 볼 수 있을 거예요. 윈도우 탐색기에서 확인해도 되지만, 아래와 같이 파이썬 셸에서 **os.path.exists()**로 확인할 수도 있어요.

```
>>> import os
>>> os.path.exists('users')
True
```

이제 이 파일 내용을 파이썬에서 읽어들여 볼까요?

```
>>> f = open('users', 'rb')
```

```
>>> a = pickle.load(f)
>>> print(a)
{'sun80': '393948', 'kim': '3kid9', 'ljm': 'py90390'}
```

사실 방금 보여드린 것은 그리 복잡할 것도 없지만, `pickle` 모듈은 파이썬에서 만들어지는 것은 뭐든지 다 파일에 적을 수 있다고 합니다.

피클은 이쯤 해두고, 전에 잠깐 구경했던 `glob` 모듈을 알아보도록 하죠.

glob

`glob` 는 파일들의 리스트를 뽑을 때 사용하는데, 파일의 경로명을 이용해서 입맛대로 요리할 수 있답니다.

```
>>> from glob import glob
>>> glob('*.*')
['python.exe', 'pythonw.exe']
>>> glob('*.txt')
['LICENSE.txt', 'NEWS.txt']
```

위의 `glob()` 함수는 인자로 받은 패턴과 이름이 일치하는 모든 파일과 디렉터리의 리스트를 반환합니다. 패턴을 그냥 *라고 주면 모든 파일과 디렉터리를 볼 수 있어요.

현재 경로가 아닌 다른 경로에 대해서도 조회할 수 있습니다.

```
>>> glob(r'C:\U*')
# C:\에서 이름이 U로 시작하는 디렉터리나 파일을 찾기
['C:\\Users', 'C:\\\\usr']
```

위에서는 `r`로 시작하는 원시 (raw) 문자열을 사용한 것에 유의하세요.

os.path

다음은 `glob` 과 함께 `os.path` 모듈을 사용한 예제입니다.

```
from glob import glob
from os.path import isdir

for x in glob('*'):
    if isdir(x):
        print(x, '<DIR>')
    else:
        print(x)
```

어떤 일을 하는 코드인지 짐작이 가시는지요?

`glob('*)`을 사용해 얻은 리스트의 원소 x를 하나씩 출력하되, 그것이 디렉터리이면 <DIR>이라는 문자열을 뒤에 붙여서 출력하게 했답니다.

실행 결과는 다음과 같습니다.

```
DLLs <DIR>
Doc <DIR>
img_read.py
include <DIR>
Lib <DIR>
libs <DIR>
LICENSE.txt
NEWS.txt
python.exe
python3.dll
python38.dll
pythonw.exe
Scripts <DIR>
tcl <DIR>
Tools <DIR>
vcruntime140.dll
```

6.3.1 연습 문제: 애국가

문제

[korean_national_anthem_1.txt](#) 파일에는 애국가 1 절 가사가 있습니다.

```
동해 물과 백두산이 마르고 닳도록  
하느님이 보우하사 우리나라 만세.  
무궁화 삼천리 화려 강산  
대한 사람, 대한으로 길이 보전하세요.
```

마찬가지로 4 절까지의 가사가 텍스트 파일에 나뉘어 저장되어 있습니다.

- [korean_national_anthem_2.txt](#)
- [korean_national_anthem_3.txt](#)
- [korean_national_anthem_4.txt](#)

이 파일들을 읽어 `out.txt` 파일에 다음과 같이 저장하는 파이썬 스크립트를 작성하세요.

```
korean_national_anthem_1.txt  
-----  
동해 물과 백두산이 마르고 닳도록  
하느님이 보우하사 우리나라 만세.  
무궁화 삼천리 화려 강산  
대한 사람, 대한으로 길이 보전하세요.  
  
korean_national_anthem_2.txt  
-----  
남산 위에 저 소나무, 칠갑을 두른 듯  
바람 서리 불변함은 우리 기상일세.  
무궁화 삼천리 화려 강산  
대한 사람, 대한으로 길이 보전하세요.  
  
korean_national_anthem_3.txt  
-----  
가을 하늘 공활한데 높고 구름 없이  
밝은 달은 우리 가슴 일편단심 일세.  
무궁화 삼천리 화려 강산
```

```
대한 사람, 대한으로 길이 보전하세.
```

```
korean_national_anthem_4.txt
```

```
-----  
이 기상과 이 맘으로 충성을 다하여  
괴로우나 즐거우나 나라 사랑하세.  
무궁화 삼천리 화려 강산  
대한 사람, 대한으로 길이 보전하세.
```

답

- 코드: [ch06/text_file_merge.py](#)

6.4 응용 예제: 음성 인식을 활용한 일본어 퀴즈

요즘 일본어 공부를 다시 시작했습니다. 옛날에 잠깐 배운 적이 있지만 너무 오래 돼서 하라가나도 잊어버렸어요. ㅠ

그래서 일본어 공부를 도와주는 프로그램을 파이썬으로 만들어 보려고 합니다.

구현하려는 기능은 단순합니다.

화면에 일본어로 출력된 단어를 읽으면, 마이크로 소리를 받아서 맞게 읽었는지 알려주는 것이죠.

음성 (speech) 을 텍스트 (text) 로 바꿔주는 기능을 구현할 수 있다면 [영어 퀴즈](#) 연습 문제와도 크게 다르지 않습니다.

파이썬으로 음성을 텍스트로 바꾸는 방법을 검색해보니 [SpeechRecognition](#)이라는 패키지가 있네요. 이를 설치하고 간단히 테스트해보겠습니다.

준비

아나콘다 가상환경 생성

저는 아나콘다 가상 환경을 만들어서 작업했습니다. 아나콘다 프롬프트에서 다음 명령을 실행합니다.

```
conda create -n japanese_study Python=3.8  
conda activate japanese_study
```

SpeechRecognition 설치

pip로 SpeechRecognition 패키지를 설치합니다.

```
pip install SpeechRecognition
```

PyAudio 설치

SpeechRecognition에서 마이크를 사용하려면 PyAudio도 필요합니다.

Windows에는 다음 명령으로 설치합니다.

```
pip install pipwin  
pipwin install pyaudio
```

macOS에는 다음 명령으로 설치합니다. FLAC도 함께 설치합니다.

```
xcode-select --install  
brew remove portaudio  
brew install portaudio  
brew install flac  
pip install pyaudio
```

마이크 테스트

파이썬 셸을 실행하고, 앞에서 설치한 패키지들이 잘 작동하는지 확인합니다.

```
>>> import speech_recognition as sr  
>>> sr.__version__  
'3.7.1'  
>>> r = sr.Recognizer()  
>>> mic = sr.Microphone()
```

영어:

```
>>> with mic as source:  
...     audio = r.listen(source)  
...  
>>> r.recognize_google(audio)  
'hello how are you'
```

영어가 잘 인식된다면 한국어와 일본어도 별 문제 없이 잘될 거예요.

한국어:

```
>>> with mic as source:  
...     audio = r.listen(source)  
...  
>>> r.recognize_google(audio, language="ko-KR")  
'아 아 마이크 테스트 하나 둘 셋'
```

일본어:

```
>>> with mic as source:  
...     audio = r.listen(source)  
...  
>>> r.recognize_google(audio, language="ja")  
'おはようございます'
```

프로그램 작성

여기까지 잘 되는 것을 확인했으니 이제 프로그램을 작성해보겠습니다.

데이터 파일

데이터 파일은 [와세다 대학교 일본어 기초 수업 영상](#)을 참고해서 제가 직접 작성했습니다. 처음에는 히라가나와 영어 발음만 넣었는데, 나중에 테스트하다보니 음성인식 결과는 주로 한자로 나오는 것을 알게 되었습니다. 그래서 [히라가나,한자어,영어발음](#)의 데이터 형태가 되었습니다.

파일: [ch06/japanese_word.csv](#)

| | | | |
|---|-----|----|--------|
| 1 | あい | 愛 | ai |
| 2 | えき | 駅 | eki |
| 3 | いえ | 家 | ie |
| 4 | ここ | 此処 | koko |
| 5 | うえ | 上 | ue |
| 6 | かお | 顔 | kao |
| 7 | あした | 明日 | ashita |
| 8 | すし | 寿司 | sushi |

파이썬 코드

코드는 다음과 같습니다.

코드: [ch06/japanese_quiz.py](#)

```
import random
import time

import speech_recognition as sr

f = open('japanese_words.csv', encoding='utf-8')
lines = f.readlines()
random.shuffle(lines)

r = sr.Recognizer()
mic = sr.Microphone()

for line in lines:
    hiragana, kanji, pronunciation = line.split(',')
    correct = None

    # 틀리면 맞힐 때까지 같은 단어를 반복
    while not correct:
        if correct is None:
            print('Read this word!:')
        else:
            print('Let\'s try again!')

        print(hiragana)

        # 답을 생각하느라 머뭇거리고 있으면 음성 인식이 종료되어 버려서
        # 엔터 키 입력 후부터 인식하게 하려고 넣었습니다.
        input('Press Enter when you ready and then read it ...')

        with mic as source:
            audio = r.listen(source)

        a = r.recognize_google(audio, language='ja')
        print('Your answer:', a)

        if kanji == a:
            correct = True
            print('You\'re right!')
        else:
            correct = False
            print(hiragana, 'is pronounced', pronunciation)

    print('-' * 20)
    time.sleep(2)
```

실행 결과:

```
(japanese_study) C:\wikidocs-chobo-python\ch06>python japanese_quiz.py
Read this word!:
さかな
```

```
Press Enter when you ready and then read it ...
Your answer: たかな
さかな is pronounced sakana

Let's try again:
さかな
Press Enter when you ready and then read it ...
Your answer: 魚
You're right!
-----
Read this word!:
あさ
Press Enter when you ready and then read it ...
Your answer: 羽田
あさ is pronounced asa

Let's try again:
あさ
Press Enter when you ready and then read it ...
Your answer: 羽田
あさ is pronounced asa

Let's try again:
あさ
Press Enter when you ready and then read it ...
Your answer: 朝
You're right!
-----
Read this word!:
かお
Press Enter when you ready and then read it ...
Process finished with exit code -1
```

- 테스트 영상: <https://youtu.be/AqWlbhSk7A8>

참고

다음 문서를 참고했습니다.

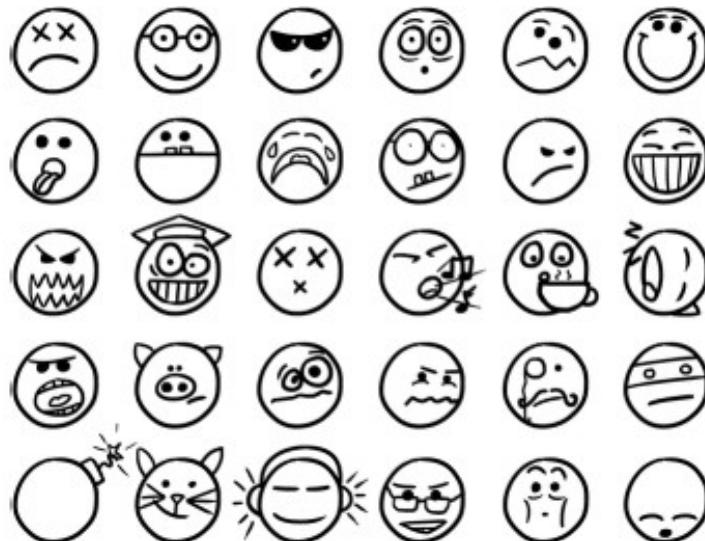
- <https://realpython.com/python-speech-recognition/>
- <https://stackoverflow.com/a/59048571/1558946>

7. 객체지향

객체지향 (Object-Oriented) 프로그래밍을 파이썬에서는 어떻게 할 수 있는지 알아봅시다.

이 장에서 배우는 것:

- 클래스와 인스턴스
- 변수와 메서드
- 상속
- 객체 속의 객체
- 특별한 메서드들



7.1. 클래스 (class) 와 인스턴스

객체지향 개념이 나타나기 이전의 프로그래밍 방법에서는 프로그램이 어떤 일을 하고 나서, 그다음엔 어떤 일을 하고, 또 그다음엔 뭘 하라는 식으로 컴퓨터가 해야 할 일을 알려주기에 바빴습니다.

그런데 객체지향 프로그래밍 (Object-Oriented Programming)에서는 프로그램을 작성할 대상이 되는 실제 세계의 사물 (객체) 을 그대로 표현하고, 그것들이 어떻게 움직이는지 정해주고 나서야 비로소 그 객체들에게 일을 시킵니다. 객체지향 프로그래밍을 잘 사용하면 보다 좋은 프로그램을 빨리 만들 수 있고, 나중에 수정하기도 편해진다고 합니다. 프로그래밍 언어들이 모두 객체지향적인 것은 아니지만, 요즘에 널리 사용되는 언어 중에는 객체지향을 지원하는 것이 많습니다.

파이썬은 꼭 객체지향적으로 작성하지 않아도 됩니다. 다시 말씀드려서, 앞으로의 강좌를 보지 않으셔도 간단한 프로그램을 작성하는데 문제가 없다는 것이죠.

하지만 객체지향에 대해 이해하시고 나면 파이썬으로 윈도우 프로그래밍을 하거나, 복잡한 프로그램을 작성하는데 많은 도움이 된답니다. 또, 다른 프로그래밍 언어를 이해하기도 수월해지지요.

그럼 함께 시작해보실까요?

클래스? 인스턴스?

‘김연아’ 는 실제로 존재하죠? 네, 여러분이 생각하시는 그 김연아 맞아요. ㅎㅎ ‘김동성’ 도 실제로 존재하죠? 두 사람 다 실제로 존재하는 사람입니다.

두 사람의 공통점은 무엇일까요? 여러 가지를 들 수 있겠지만 둘 다 ‘스케이터’ 라는 공통점을 갖고 있지요.

‘스케이터’ 라는 단 하나의 사람이나 물건이 실제로 존재할까요? 그렇지는 않습니다. 하지만 우리는 ‘스케이트 타는 사람’ 을 ‘스케이터’ 라고 부릅니다. 이런 것을 일컫는 말이 클래스 (class) 입니다. 우리말로 옮기기는 쉽지 않지만 ‘부류’ 라는 의미로 생각하시면 좋을 것 같아요.

다른 예를 들어볼까요?

‘사과’는 클래스이구요, ‘내가 엊저녁에 먹은 사과 다섯 개 중에 두 번째 것’이라고 콕 찍어서 말해주면 실체 (instance)로 봐줄만합니다.

‘좋은 집’은 실체일까요? 어느 한 집만을 콕 찍어서 ‘좋은 집’이라고 하기는 힘들 것 같군요. 그럼 ‘우리 집’은 실체일까요? 그건 실체라고 해도 될 것 같네요. 단, 집을 여러 채 가진 사람이 ‘우리 집’이라고 말할 때는 정확히 어느 집을 가리키는 것인지 알 수 없겠죠. 프로그램 작성是为了 클래스를 설계하다보면 이런 애매한 문제를 만날 때도 있지요.

파이썬의 클래스

이제 파이썬으로 부류와 실체를 표현해보도록 하겠습니다.

```
>>> class Singer:                      # 가수를 정의하겠느니라...
...     def sing(self):                 # 노래하기 메서드
...         return "Lalala~"
...
>>> taeji = Singer()                  # 태지를 만들어랏!
>>> taeji.sing()                     # 노래 한 곡 부탁해요~
'Lalala~'
```

클래스를 만들 때는 위와 같이 **class** 클래스이름: 형식으로 시작해서 그 다음부터 그 클래스의 성질이나 행동을 정의해주면 됩니다. 둘째 줄에는 함수가 정의되어 있죠? 이와 같이 클래스 내부에 정의된 함수를 메서드 (method)라고 부릅니다.

여기서 `sing` 메서드는 `Singer`라는 클래스가 하는 행동을 정의하고 있죠. `Singer` 클래스를 만든 다음에는 `taeji`라는 객체를 만들었습니다. 인스턴스명= 클래스()와 같이 만들면 되죠.

그 다음엔 그렇게 만들어진 `taeji`에게 노래를 시켜봤습니다. `Singer` 클래스에 `sing` 메서드를 정의해줬기 때문에 `Singer` 클래스에 속한 `taeji` 객체도 `sing` 메서드를 사용할 수 있지요. 다시 말해서 가수는 노래할 수 있으니까 태지라는 가수도 역시 노래를 할 수 있는 것입니다. 이와 같이 어떤 객체의 메서드를 사용할 때는 객체.메서드 형식으로 해주시면 됩니다.

이번엔 같은 방법으로 리키 마틴 객체를 만들어서 노래를 청해보세요.

```
>>> ricky = Singer()
>>> ricky.sing()
'Lalala~'
```

다른 클래스들도 한번씩 만들어보시기 바랍니다.

7.2. 변수와 메서드

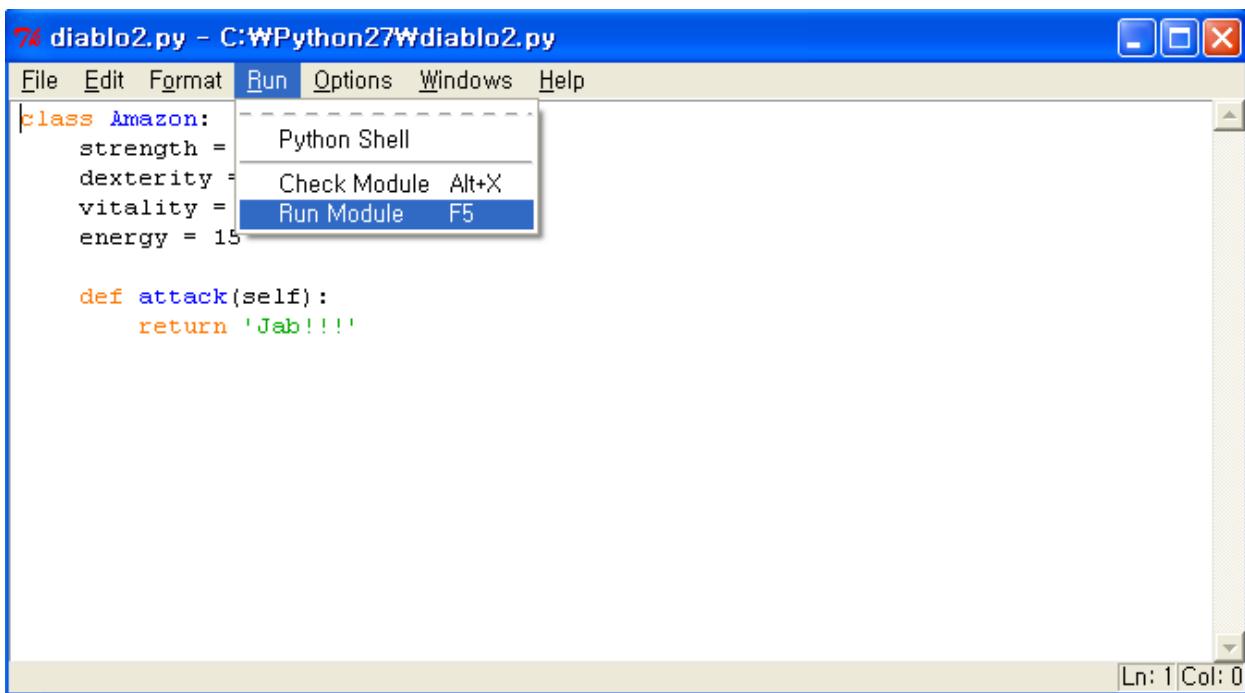
지난 시간에 클래스와 객체가 무엇인지 잠깐 살펴보았죠? 실제 세계에 존재하는 실체 (instance) 를 객체 (object) 라고 하고, 객체들의 공통점을 간추려서 개념적으로 나타낸 것이 클래스 (class) 라고 했습니다.

어떤 클래스를 만들려면 그 객체가 갖는 성질과 그 객체가 하는 행동을 정의해주면 된다고도 했고요. 디아블로 2 게임의 아마존이라는 캐릭터를 클래스로 표현해 볼까요?

```
class Amazon:
    strength = 20
    dexterity = 25
    vitality = 20
    energy = 15

    def attack(self):
        return 'Jab!!!'
```

아마존 클래스가 갖고 있는 힘, 기술, 체력, 에너지라는 네 가지 성질은 변수로 나타내었구요, '공격' 하는 행동은 메서드로 나타내었습니다. IDLE(Python GUI)에서 File - New Window 메뉴를 선택하여 새 창을 띄우고 위의 코드를 작성하여 diablo2.py 라는 파일로 저장해주세요.



F5 키를 눌러 모듈을 실행한 다음, Shell 창으로 되돌아가서 diablo2 모듈을 임포트한 다음, Amazon 클래스의 객체를 만들어 봅시다.

```
>>> import diablo2
>>> jane = diablo2.Amazon()
>>> mary = diablo2.Amazon()
```

두 명의 여전사가 탄생했습니다. 짜잔 ~

jane 과 mary 는 둘 다 Amazon 으로서 필요한 자질을 모두 갖추고 있겠죠? 그렇다면 jane 의 힘도, 공격하는 행동도 Amazon 클래스에서 정의한 그대로이겠구요.

```
>>> jane.strength
20
>>> jane.attack()
'Jab!!!'
```

이렇게 객체는 클래스에서 정의해준 변수와 메서드를 그대로 갖게 됩니다. 별로 어렵지 않죠? 여기서 주의해서 보실 것은 메서드를 정의할 때와 사용할 때는 차이가 있다는 점입니다. Amazon 클래스에서 메서드를 정의할 때는 def attack(self): 와 같이 self 라는 인자를 받았는데, jane 객체의 메서드를 호출할 때는 그냥 attack() 이라고 했지요?

IDLE에서 Run Module 안 될 때 해결 방법 <https://yong-it.blogspot.com/2018/10/python-idle.html>

self

self라는 것은 바로 그 클래스의 객체를 가리키는데, jane과 mary가 똑같은 attack 메서드를 가지기 때문에 서로 구별하기 위해서 사용한 것입니다. 한마디로, 메서드를 정의할 때는 항상 self라는 인자를 써준다고 생각하시면 되겠네요. self를 어떻게 사용하는지 좀 더 살펴보기 위해 Amazon 클래스에 메서드를 추가해 봅시다.

```
def exercise(self):
    self.strength += 2
    self.dexterity += 3
    self.vitality += 1
```

이 메서드는 훈련을 하면 힘, 기술, 체력이 올라가는 것을 표현했지요. diablo2.py 파일의 Amazon 클래스에 이 메서드를 추가하고 저장해주시기 바랍니다. 저장하셨으면 F5 키를 다시 한번 눌러서 diablo2 모듈을 다시 실행하고, 임포트도 해준 다음에, 새로운 객체를 만들어서 훈련을 시켜보세요.

```
>>> import diablo2
>>> eve = diablo2.Amazon()
>>> eve.exercise()
>>> eve.strength
22
```

힘(strength)이 세졌네요. 훈련한 보람이 있지요?

7.3. 상속

지난 두 강좌를 통해 객체 지향에 대해 소개해 드렸는데, 여러분은 어떻게 느끼셨는지요. 개념적으로는 이해가 될 듯도 한데, 실제로 프로그램을 작성할 때는 어떻게 적용해야 할지 난감해하시는 분이 많지 않을까 하는 생각도 드는군요.

사실 객체지향의 개념을 제대로 이해하고 활용하기까지는 시간이 좀 걸린답니다. 그러니까 너무 걱정하지 마시고 천천히, 여러 번 반복해서 공부하시기 바랍니다.

아무리 좋은 개념도 갑자기 많이 공부하면 오히려 헷갈리기 쉽지요. 그러니 제가 설명드리는 것부터 프로그램에 적용시켜 보면서 차근차근 공부하셨으면 합니다.

오늘 알아볼 내용은 객체지향 프로그래밍의 핵심적인 개념 가운데 하나인 상속 (inheritance)입니다. 상속이란 어떤 클래스가 다른 클래스의 성질을 물려받는 것을 말하지요.

어떤 클래스를 만들 때 처음부터 모든 것을 새로 만들 필요 없이, 핵심적인 성질을 갖고 있는 다른 클래스로부터 상속을 받아서 조금만 손을 보면 쓸만한 클래스를 만들 수 있답니다.

예제를 보실까요?

```
class Person:  
    # 눈 두 개, 코 하나, 입 하나...  
    eyes = 2  
    nose = 1  
    mouth = 1  
    ears = 2  
    arms = 2  
    legs = 2  
  
    # 먹고 자고 이야기하고...  
    def eat(self):  
        print('암남...')  
  
    def sleep(self):  
        print('쿨쿨...')  
  
    def talk(self):  
        print('주절주절...')
```

위의 Person이라는 클래스는 보통 사람을 나타낸 클래스입니다. 눈, 코, 입, 팔다리가 다 있고, 먹고, 자고 이야기도 하지요.

이번에는 학생이라는 클래스를 만들어 봅시다. 학생도 사람니까 사람이 갖는 여러 성질이나 행동은 모두 갖고 있을 것이고, 거기에 학생만의 특징을 좀 더 갖도록 하면 되겠지요? 하지만, 사람 클래스도 한참 걸려서 입력했는데, 또다시 학생클래스의 눈, 코, 입부터 시작해서 모든 것을 새로 만들어주려면 너무 귀찮겠네요.

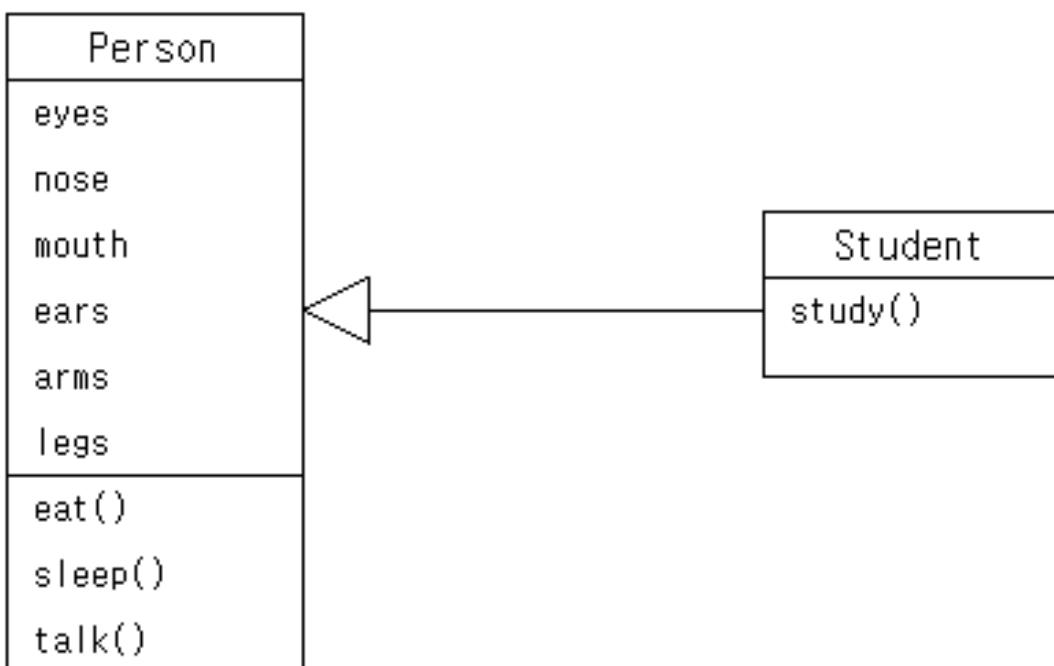
바로 이럴 때 다음과 같이 상속을 이용하면 손쉽게 학생 클래스를 만들 수 있답니다.

```
class Student(Person):
    def study(self):
        print('열공 열공...')
```

위의 Student 클래스는 Person 클래스를 상속받았습니다. 우리가 Student 클래스를 눈, 코, 입부터 하나하나 다시 만들어 주지 않더라도 Person의 성질들을 모두 물려받아서 갖게 된 것이죠. 우리는 여기에 study라는 메쏘드만 하나 더 써주어서 우아하게 마무리를 했습니다.

굳이 상속을 받지말고 스크립트를 복사해서 붙이면 되지 않느냐고요? 물론 그렇게 해도 가능합니다.

하지만 나중에 사람과 학생 클래스에 ‘웃 색깔’이라든지, ‘싸우다’ 같은 것들을 추가하고 싶어진다면, 그 때마다 사람 클래스와 학생 클래스를 각각 수정해야 되겠지요.



사람과 학생의 관계를 위와 같이 그림으로 표현할 수도 있습니다. 사각형은 각각의 클래스를 나타내고,

그 안에 클래스의 이름과 변수, 메쏘드를 적어주었습니다. 화살표는 '상속 관계'를 나타내고, 그 방향은 하위 클래스 (상속받은 클래스)로부터 상위 클래스 (상속해준 클래스)를 향하고 있습니다.

이 화살표를 따라가면서 'is a' 라고 읽으면 두 클래스의 관계를 쉽게 파악할 수 있지요.

"A Student is a Person.(학생은 사람이다)" 이 되는군요.

클래스들의 관계를 이렇게 그림으로 그리면 클래스를 설계하거나 분석할 때 이해하기 쉽답니다.

그럼, 과연 Student 클래스가 Person 클래스의 모든 성질을 똑같이 갖고 있는 것인지도 확인해 보도록 하죠. 먼저 위의 예제들을 파일로 저장해서 import 하시거나, 그냥 인터프리터에서 입력하신 다음에 아래와 같이 테스트를 해보세요.

```
>>> lee = Person()
>>> lee.mouth
1
>>> lee.talk()
주 절 주 절...
>>> kim = Student()
>>> kim.mouth
1
>>> kim.talk()
주 절 주 절...
```

Person 클래스의 객체인 lee 와 Student 클래스의 객체 kim 이 하는 짓들이 똑같지요? Person 클래스로부터 상속받았기 때문에 그렇다는 것을 아실 수 있겠죠? 그러나, 상속 받은 것으로 끝이 아니지요... Student 는 공부라는 비장의 카드도 갖고 있지 않았겠습니까?

```
>>> kim.study()
열 공 열 공...
```

역시 kim 은 학생답게 공부도 열심히 하는군요...

7.3.1 메서드 상속과 재정의

이번에는 동물을 나타내는 클래스들을 만들어 볼게요.

코드: [ch07/animal.py](#)

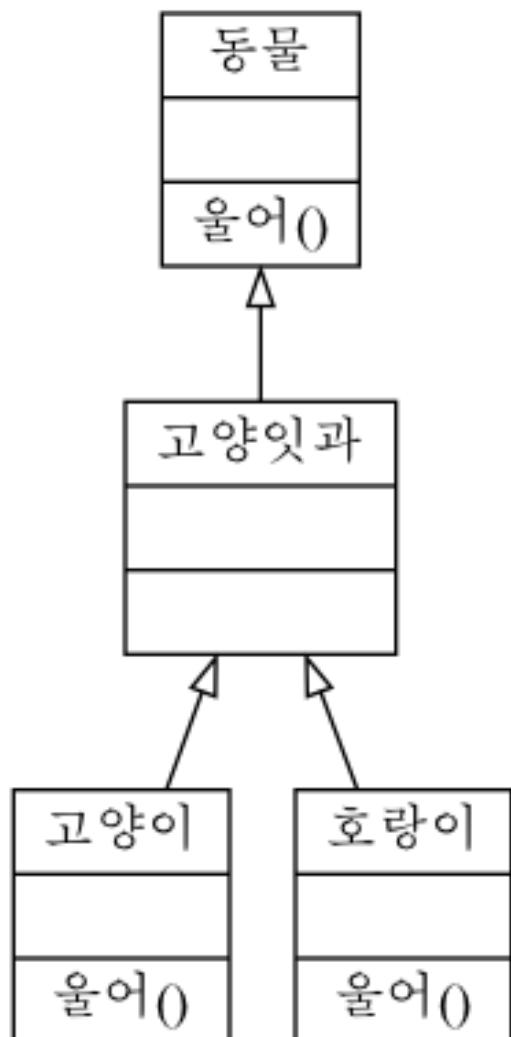
```
class 동물:
    def 울어(self):
        print('...')

class 고양잇과(동물):
    pass

class 호랑이(고양잇과):
    def 울어(self):
        print('어흥')

class 고양이(고양잇과):
    def 울어(self):
        print('야옹')
```

이 클래스들의 상속 관계를 그림으로 나타내면 이렇습니다.



animal.py 모듈이 있는 곳에서 파이썬 셸을 열어서 임포트합니다.

```
>>> import animal
```

나비라는 이름의 고양이를 만듭니다. (새로운 고양이 인스턴스를 만들므로 팔호를 붙여야 합니다.)

```
>>> 나비 = animal.고양이()
```

나비는 야옹하고 울어요.

```
>>> 나비.울어()  
야옹
```

이번에는 호돌이라는 호랑이를 만들어 볼까요?

```
>>> 호돌이 = animal.호랑이()
>>> 호돌이.울어()
어흥
```

수호랑이라는 이름의 호랑이도 만들어 보세요.

isinstance()

위에서 만든 나비는 고양이 클래스의 인스턴스이고, 호랑이와 수호랑은 호랑이 클래스의 인스턴스입니다.

파이썬의 내장 함수인 **isinstance()**를 사용하면, 객체가 특정 클래스의 인스턴스인지 확인할 수 있어요. (이때는 인스턴스를 새로 만드는 것이 아니므로 `animal.고양이` 뒤에 괄호를 붙이지 않습니다.)

```
>>> isinstance(나비, animal.고양이)
True
>>> isinstance(나비, animal.호랑이)
False
```

그런데 고양이는 고양잇과를 상속한 클래스이고, 고양잇과는 동물 클래스를 상속한 클래스였죠? 따라서 나비는 고양이 클래스의 인스턴스인 동시에, 고양잇과 클래스와 동물 클래스의 인스턴스입니다.

```
>>> isinstance(나비, animal.고양잇과)
True
>>> isinstance(나비, animal.동물)
True
```

issubclass()

A라는 클래스가 B라는 클래스의 하위 클래스 (subclass)인지, 즉 A가 B를 상속했는지 알려면 **issubclass()**라는 내장 함수를 쓰면 됩니다.

앞에서 만든 고양이 클래스가 고양잇과와 동물 클래스의 하위 클래스인지 확인해 볼까요?

```
>>> issubclass(animal.고양이, animal.고양잇과)
True
>>> issubclass(animal.고양이, animal.동물)
True
```

울어 () 메서드의 상속과 재정의

앞에서 동물 클래스에는 울어()라는 메서드가 있었는데, 고양잇과 클래스에는 메서드를 따로 만들지 않고 그냥 넘어갔었죠 (**pass**).

보비라는 푸마가 있다고 합니다. 푸마 클래스가 따로 없으니 고양잇과 클래스의 인스턴스로 만들어 줄게요.

```
>>> 보비 = animal.고양잇과()
```

고양잇과 클래스에는 울어() 메서드를 정의하지 않았는데, 보비는 울 수 있을까요?

```
>>> 보비.울어()  
...
```

고양잇과가 동물을 상속했으므로 울어() 메서드를 따로 만들어 주지 않았음에도 ...이 출력되는 걸 볼 수 있습니다.

그런데 고양이와 호랑이에서는 동물의 울어()를 그대로 쓰지 않고 새로 정의해서 각기 다른 울음 소리를 냈었죠? 이와 같이 **메서드를 재정의**할 수도 있습니다.

7.4. 객체 속의 객체

코끼리를 냉장고에 넣는 방법을 아시나요? 아마 모르시는 분이 없겠죠? 제가 아는 유머는 온 국민이 다 알고 계시니까요.

1번, 냉장고 문을 연다.

2번, 코끼리를 넣는다.

3번, 냉장고 문을 닫는다. ^^;

이걸로 프로그램을 한번 짜볼까요?

```
class Fridge:
    def __init__(self):
        self.isOpened = False
        self.foods = []

    def open(self):
        self.isOpened = True
        print('냉장고 문을 열었어요...')

    def put(self, thing):
        if self.isOpened:
            self.foods.append(thing)
            print('냉장고 속에 음식이 들어갔네요...')
        else:
            print('냉장고 문이 닫혀 있어서 못 넣겠어요...')

    def close(self):
        self.isOpened = False
        print('냉장고 문을 닫았어요...')

class Food:
    pass
```

위와 같이 냉장고와 음식 클래스를 갖고 있는 `fridge.py` 모듈을 만들어보았습니다. 냉장고 클래스에는 문이 열려있는지를 나타내는 `isOpened`라는 변수와 냉장고 안에 들어있는 음식들의 리스트인 `foods`가 있습니다. 또, 냉장고 문을 열고, 음식을 집어넣고, 문을 닫는 메서드도 각각 갖고 있지요.

음식에 대해서는 별로 쓸 말이 없더군요. 쓸 것이 없을 때는 `pass` 라고만 써주면 된다고 해서 그렇게 했습니다. 속이 빈 음식 클래스를 만든 거지요. 이제 인터프리터를 띄워서 냉장고에다가 코끼리를 집어넣어 봅시다.

```
>>> import fridge  
>>> f = fridge.Fridge()  
>>> apple = fridge.Food()  
>>> elephant = fridge.Food()
```

먼저 냉장고 클래스의 객체로 `f`라는 것을 만들고 음식 클래스의 객체는 `apple` 과 `elephant`를 만들었습니다.

```
>>> f.open()  
냉장고 문을 열었어요...  
>>> f.put(apple)  
냉장고 속에 음식이 들어갔네요...
```

냉장고 문을 열고, 일단 준비운동으로 냉장고에 사과를 넣어봤는데 잘 들어갔지요? 그럼 코끼리도 넣어 볼까요?

```
>>> f.put(elephant)  
냉장고 속에 음식이 들어갔네요...
```

코끼리도 쑥 들어갔습니다 ~ 냉장고 속에 사과랑 코끼리가 잘 들어갔는지 확인해볼까요? 냉장고 `f`의 `foods` 리스트에 뭐가 들어있는지 봅시다.

```
>>> f.foods  
[<fridge.Food instance at 007924AC>, <fridge.Food instance at 0079153C>]
```

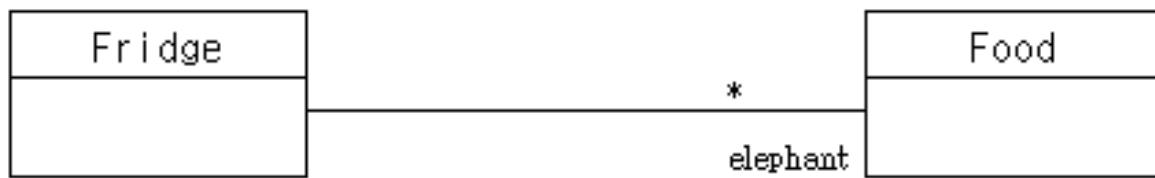
`Food` 클래스의 인스턴스 (instance, 실체) 두 개가 들어있다고 나오는군요. 실체나 객체나 비슷한 말이겠죠? 자, 냉장고 객체는 `foods`라는 리스트를 갖고 있구요, `foods` 리스트는 음식 클래스의 사과와 코끼리 객체를 갖고 있습니다. 결국 냉장고 객체는 다른 객체들을 갖고 있다고도 할 수 있겠죠?

오늘 보신 것처럼 객체는 또 다른 객체를 포함할 수도 있답니다. 객체지향 프로그래밍에서는 이런 것을 composition(합성, 복합) 이라고 하구요, ‘has-a’ 관계라고도 합니다.

“`f`는 `elephant`를 갖고 있다 (`f has an elephant`).” 말 되죠?

별로 어렵지는 않지만 잘 써먹을 수 있는 개념이니까 익혀두세요.

이렇게해서 오늘의 임무인 객체 속에 객체 집어넣기는 훌륭하게 완수했군요.



참고로 ‘has-a’ 관계를 나타낸 그림을 보여드리면서 마치도록 하겠습니다. Fridge 클래스의 객체는 Food 클래스의 객체 (elephant 등...) 를 여러 개 (*) 가질 수 있다는 것을 연결선 위아래에 *와 elephant라고 써줘서 나타낸 것이지요.

7.5. 특별한 메서드들

이제 메서드에 대해서는 다들 알고 계시겠죠? 메서드라는 것은 우리가 클래스를 만들면서 그 안에 만들어 넣은 함수를 말하지요? 만들어진 메서드를 사용하려면 `객체.메서드()`와 같은 형식으로 호출을 해주었고요. 오늘은 그런 일반적인 메서드들과는 조금 다른 특별한 메서드들에 대해 함께 알아보려고 합니다.

`__init__` 메서드 (초기화)

```
# bookstore.py

class Book:

    def setData(self, title, price, author):
        self.title = title
        self.price = price
        self.author = author

    def printData(self):
        print('제목 : ', self.title)
        print('가격 : ', self.price)
        print('저자 : ', self.author)

    def __init__(self):
        print('책 객체를 새로 만들었어요~')
```

예제로 Book(책) 클래스를 갖는 bookstore(책방) 모듈을 만들어 보았습니다.

책 클래스의 메서드로는 책 제목, 가격, 저자와 같은 자료들을 입력할 때 사용할 `setData()` 와 이런 자료들을 출력해주는 `printData()` 를 만들어 주었지요.

그리고 `__init__` 이라는 메서드도 있지요? 이것이 바로 파이썬에서 특별하게 약속된 메서드 가운데 하나로, 초기화 (initialize) 메서드라고도 합니다.

어떤 클래스의 객체가 만들어질 때 자동으로 호출되어서 그 객체가 갖게 될 여러 가지 성질을 정해주는 일을 하지요. 그럼 책 클래스의 객체를 하나 만들어볼까요?

```
>>> import bookstore  
>>> b = bookstore.Book()  
책 객체를 새로 만들었어요~
```

Book() 해서 Book 객체를 만들자마자 초기화 메서드가 실행되었군요. 나머지 setData 와 printData 메서드들은 다음과 같이 사용하시면 됩니다.

```
>>> b.setData('누가 내 치즈를 먹었을까', '300원', '미키')  
>>> b.printData()  
제목 : 누가 내 치즈를 먹었을까  
가격 : 300원  
저자 : 미키
```

이제 초기화 메서드가 뭔지 대충 감을 잡으셨으면 __init__ 메서드를 사용해서 실제로 객체를 초기화 시켜보겠습니다. __init__ 메서드를 아래와 같이 수정해보세요.

```
def __init__(self, title, price, author):  
    self.setData(title, price, author)  
    print('책 객체를 새로 만들었어요~')
```

객체를 생성시킬 때 제목, 가격, 저자를 인자로 받아서, setData 메서드에게 넘겨주도록 했죠? 물론 초기화 메서드에서 직접 변수를 다뤄도 상관없지만 setData 메서드를 미리 만들어뒀으니까 이용을 한 것입니다. 이제부터 책 객체를 만들 때는 다음과 같이 세 개의 인자를 넘겨줘야합니다.

```
>>> from importlib import reload  
>>> reload(bookstore)  
>>> b2 = bookstore.Book('내가 먹었지롱', '200원', '미니')  
책 객체를 새로 만들었어요~
```

그런대로 쓸 만 하죠? 값이 잘 들어갔는지도 확인해보세요. 참고로 말씀드리면, 초기화 메서드와 같은 것을 다른 객체지향 언어에서는 생성자 (constructor)라고 부릅니다.

__del__ 메서드 (소멸자)

__init__ 메서드와 반대로 객체가 없어질 때 호출되는 메서드도 있습니다. 이런 것을 소멸자 (destructor)라고 하는데, 파이썬에서는 __del__ 메서드가 소멸자의 역할을 맡고 있죠.

객체가 없어지는 수도 있느냐구요? 뭐하려고 없애느냐구요? del 문을 사용해보세요. 당장 없어집니다. 또, 만 들어 둔 객체가 더 이상 필요 없어지면 파이썬이 알아서 처리해주기도 하구요.

그건 직접 `__del__` 메서드를 만들어서 테스트해보시면 잘 아실 수 있겠죠? 그냥 다른 메서드와 똑같이 작성하시면 됩니다.

어떤 객체가 없어지기 전에 뭔가 처리를 필요로 한다면 소멸자가 유용하게 쓰이겠지요?

`__repr__` 메서드 (프린팅)

이번엔 `printData` 와 같은 메서드를 호출하는 대신, 파이썬의 기본문인 `print` 문을 사용해서 책 제목을 찍어보도록 하겠습니다. 이런 일을 가능하게 해주는 것은 바로 `__repr__` 메서드이지요. 책 클래스에 아래와 같이 `__repr__` 메서드를 추가해주세요.

```
def __repr__(self):
    return self.title
```

별 다른 것은 없구요, `return` 문을 사용했다는 것만 눈여겨보시면 됩니다. `__repr__` 메서드는 ‘문자열’을 ‘`return`’한다고 생각하시면 되겠죠? 그럼 책방 모듈을 재적재하고 새 책을 만들어서 테스트해보세요.

```
>>> b3 = bookstore.Book('나두 좀 줘', '100원', '쥐벼룩')
책 객체를 새로 만들었어요~
>>> print(b3)
나두 좀 줘
```

`__add__` 메서드 (덧셈)

이제 책방은 문을 닫고 세모, 네모, 동그라미 같은 도형을 만들어볼까요?

```
# shape.py

class Shape:
    area = 0

    def __add__(self, other):
        return self.area + other.area
```

학교에서 도형에 대해 배울 때는 늘 넓이에 대해 생각을 하지요? 여기서는 두 도형의 넓이를 더하는 `__add__` 메서드를 만들어보았습니다. 두 개의 객체 `self`와 `other`를 인자로 받아서 그들의 넓이를 더한

값을 돌려주는 일을 하도록 했지요.

```
>>> a = shape.Shape()  
>>> a.area = 20  
>>> b = shape.Shape()  
>>> b.area = 10  
>>> a + b  
30
```

도형 a 와 b 를 덧셈 연산자 (+) 로 더했더니 두 도형의 넓이가 더해졌죠? 마치 보통의 두 숫자를 더하는 것처럼 간단하게 말입니다. 이와 같이 특별한 메서드를 사용해서 연산자가 하는 일을 정의하는 것을 연산자 중복 (overload) 이라고 부른답니다. 연산자 중복을 이용하면 사용자가 직접 만든 클래스의 객체에 대해서도 연산자를 쓸 수 있게 되지요. 마치 파이썬 자체에서 제공하는 자료형처럼 말입니다.

아직도 사태의 심각성을 이해하지 못하고 “a + b 가 뭐 어쨌길래? 원래 그냥 더하면 되는 거잖아 ~” 라고하시는 분들! –

Shape 클래스에 `__add__` 메서드를 넣지 말고 객체 두 개를 만든 다음에 더해보세요.

지나가던 뱀이 웃습니당...-;

그리고 벌서 눈치채신 분도 있겠지만, 덧셈 연산자 대신 `__add__` 메서드를 직접 호출해도 그 결과는 똑같답니다.

```
>>> a.__add__(b)  
30
```

그럼 도형 객체 간에 뺄셈도 할 수 있도록 `__sub__` 메서드도 만들어보세요 ~.

`__lt__` 메서드 (비교)

이제 연산자 중복에 대해 어느 정도 감이 잡히시죠? 파이썬에서 제공하는 연산자 중복 메서드는 이외에도 많기 때문에 모두 살펴보기는 힘들겠네요. 마지막으로 두 개의 객체를 비교하는 비교 연산자 (<, >, ==) 를 쓸 수 있도록 해주는 메서드를 살펴보면서 이 부분을 정리하겠습니다. 일단 Shape 클래스에 아래와 같이 `__lt__` 메서드를 추가해 주세요 ('less than' 을 의미).

```
def __lt__(self, other):  
    return self.area < other.area
```

간단히 두 객체 self 와 other 의 area 를 비교한 결과를 돌려주도록 했는데요, 이제 두 숫자를 비교하듯 두 객체를 부등호로 비교할 수 있습니다. shape 모듈을 재적재하고 새로운 객체들을 만들어서 각각 area 값 을 정해 준 다음, 두 객체를 비교해 보겠습니다.

```
>>> import shape
>>> c = shape.Shape()
>>> c.area = 30
>>> d = shape.Shape()
>>> d.area = 20
>>> c > d
True
>>> if c > d: print('c가 더 넓어요~')
...
c가 더 넓어요~
```

비교가 잘 되나요?

오늘 강좌는 좀 길어졌군요. 아까도 말씀드렸지만 연산자 중복 메서드는 오늘 보여드린 것 말고도 많이 있으니 다른 자료도 찾아보시기 바랍니다.

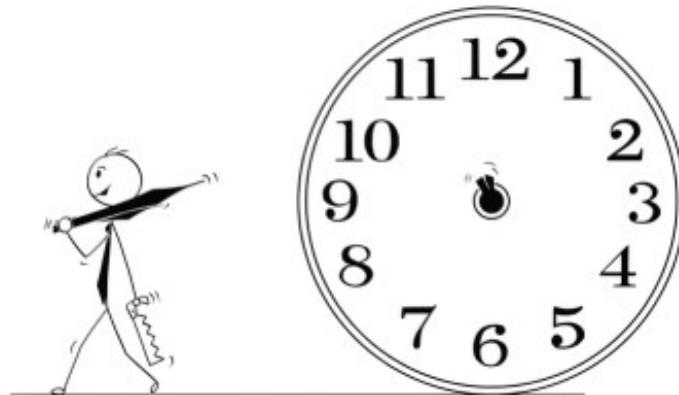
그럼... I'll be back... ^^

8. 예외

예외 (exception) 란 무엇이며 어떻게 다뤄야 할지 알아봅니다.

이 장에서 배우는 것:

- 예외 처리 (try-except)



8.1 예외처리 (try, except)

프로그래밍 언어를 배울 때에는 직접 따라해보고, 만들어보는 것이 중요합니다. 여러분도 지금까지 함께 공부하면서 연습을 많이 해보셨겠죠? 그렇다면 아래와 같은 메시지도 많이 보셨을 것 같네요.

```
>>> print 방 가~  
File "<stdin>", line 1  
    print 방 가~  
          ^  
SyntaxError: invalid syntax
```

위에선 '방 가~'라는 문자열을 출력하려고 했는데, 뭔가 문제가 생긴 것 같죠? 메시지가 난해한 것 같은데... 무슨 뜻인지 해독을 해볼까요?

```
File "<stdin>", line 1
```

파일의 1 번째 줄에서

```
print 방 가~  
          ^
```

^로 표시된 부분에

```
SyntaxError: invalid syntax
```

잘못된 구문으로 인해 오류가 발생했음

한마디로 문법이 틀렸다는 얘기군요. 여기서 이라는 건 표준 입력 (standard input), 즉 키보드를 통해 입력되는 것을 뜻합니다. 대화식으로 작성하지 않고 파일로 작성해서 실행시켰다면 그 파일의 이름이 나왔겠지요.

이렇게 파이썬은 프로그램 실행 중에 문제가 생기면, 어디가 어떻게 잘못됐는지 판단해서 우리에게 알려줍니다. 그러면 우리는 그 부분을 수정해서 잘 작동하도록 하면 되지요.

왕초보를 위한 Python: 쉽게 풀어 쓴 기초 문법과 실습

그런데, 프로그램을 짜다보면 평소에는 잘 돌아가다가 가끔씩 문제가 생기는 경우도 있답니다. 다음의 예제가 바로 그런 경우인데요, 두 수를 곱하고 나누서 더하는 함수입니다.

```
>>> def f(a, b):
...     return (a * b) + (a / b)
...
>>> f(4,2)
10
```

아직은 별다른 문제가 없어보이죠? 하지만 두 번째 인자로 0을 넘겨주면 난리가 납니다.

```
>>> f(3,0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 2, in f
ZeroDivisionError: integer division or modulo by zero
```

정수를 0으로 나누는 오류가 발생했다고 하는군요. 원래 숫자를 0으로 나눌 수가 없죠? 그렇다면 이 함수는 손을 좀 봐야겠네요.

```
>>> def f(a, b):
...     if a and b:                      # a와 b가 둘 다 0이 아닐 때
...         return (a * b) + (a / b)
...     elif a:                           # 그렇지 않고 a만 0이 아닐 때
...         return '불능'
...     else:                            # 둘 다 0일 때
...         return '부정'
...
```

이제 좀 그럴 듯 하네요.

```
>>> f(3, 0)
'불능'
>>> f(0, 0)
'부정'
```

OX 퀴즈!! 이제 이 함수는 더 이상 오류가 생길 일이 없겠죠?

그렇다구요?

과연 그럴까요...

`f(300000, 500000)`를 한번 실행시켜보세요.

계산 결과가 잘 나올 수도 있지만,

```
>>> f(3000000, 500000)
1500000000000.6
```

파이썬 버전이 낮을 경우 아래와 같이 오류가 발생하기도 합니다.

```
>>> f(300000, 500000)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 3, in f
OverflowError: integer multiplication
```

인자로 받은 두 수의 곱이 정수형으로 처리하기에는 너무 큰 값이라서 오류가 발생했군요.

음... 또 이런 건...?

```
>>> f(이십, 오)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name '이십' is not defined
```

쩝...

프로그램을 작성하다 보면 오류가 발생할 수 있는 경우는 끝도 없습니다. 우리가 이런 오류를 모두 예상해서 프로그램을 작성한다는 것은 정말로 '삽질'이라고 밖에 할 수 없겠네요. 앞의 예에서 불능과 부정을 정해준 것은 나쁘지 않다고 하더라도, 다른 두 경우까지 각각 처리해주는 건 시간 낭비이겠죠?

다행히 파이썬에선 이런 문제를 쉽게 해결해주는 방법이 있답니다. 위와 같이 프로그래머의 의도와 동떨어진 상황이 발생하는 것을 예외 (exception) 라고 해서, 예외가 발생하면 어떤 조치를 취할 것인지 정해주는 것이죠.

그럼 예외를 처리할 수 있도록 함수를 수정해 보도록 하지요.

```
>>> def f(a, b):
...     try:
...         if a and b:
...             return (a * b) + (a / b)
...     elif a:
...         return '불능'
...     else:
...         return '부정'
...     except:
...         return '똑 바로 살아라'
```

보시는 것처럼 방법은 간단합니다. 기본적인 문장들을 try 밑에 넣어주고, 예외가 발생했을 때 처리할 부

분은 `except` 밑에 넣어주면 됩니다. 일단 시도 (`try`) 해보다가 문제가 생기면 (`except`) 처리해주는 것이죠.

예외는 오류 (`error`) 보다 더 넓은 개념이긴 하지만 지금은 비슷하게 생각하셔도 되구요, C++ 과 Java 에서도 비슷하게 사용된답니다.

위의 예제는 예외가 무엇인지 설명하기 위해 간단히 작성한 것이니 참고만 하시기 바랍니다. 예외가 발생하지 않도록 막는 것이 능사가 아니라, 필요한 곳에서 적절한 예외를 발생시킬 수 있도록 궁리를 많이 해야 합니다.

더 읽을 거리

예외에 대해 더 알고 싶다면 『실용 파이썬 프로그래밍』의 ‘[오류 검사](#)’ 절을 읽어보세요.

8.2 연습 문제: 음성 인식 일본어 퀴즈 개선

문제

6장에서 응용 예제로 소개드린 **음성 인식을 활용한 일본어 퀴즈**를 풀다 보니, 프로그램이 말을 잘 알아듣지 못하고 `speech_recognition.UnknownValueError`라는 오류를 낼 때가 있네요.

다음 단어를 이어서 공부하고 싶은데 프로그램이 멈춰버리니 좀 불편합니다.

오류가 날 경우 그 단어는 통과하고 다음 단어를 공부하도록 고쳐보세요.

풀이

저는 이렇게 했어요.

- 코드: [ch08/japanese_quiz2.py](#)

저는 `except` **as** 구문을 썼는데, 어려울까봐 앞에서는 소개하지 않았어요. 궁금하신 분은 [실용 파이썬 프로그래밍 3.3 절](#)을 읽어보세요.

9. 테스팅과 성능

코드에 오류가 없는지 테스트하는 법과, 코드를 실행하는데 걸리는 시간을 측정하는 방법을 알아봅니다.

이 장에서 배우는 것:

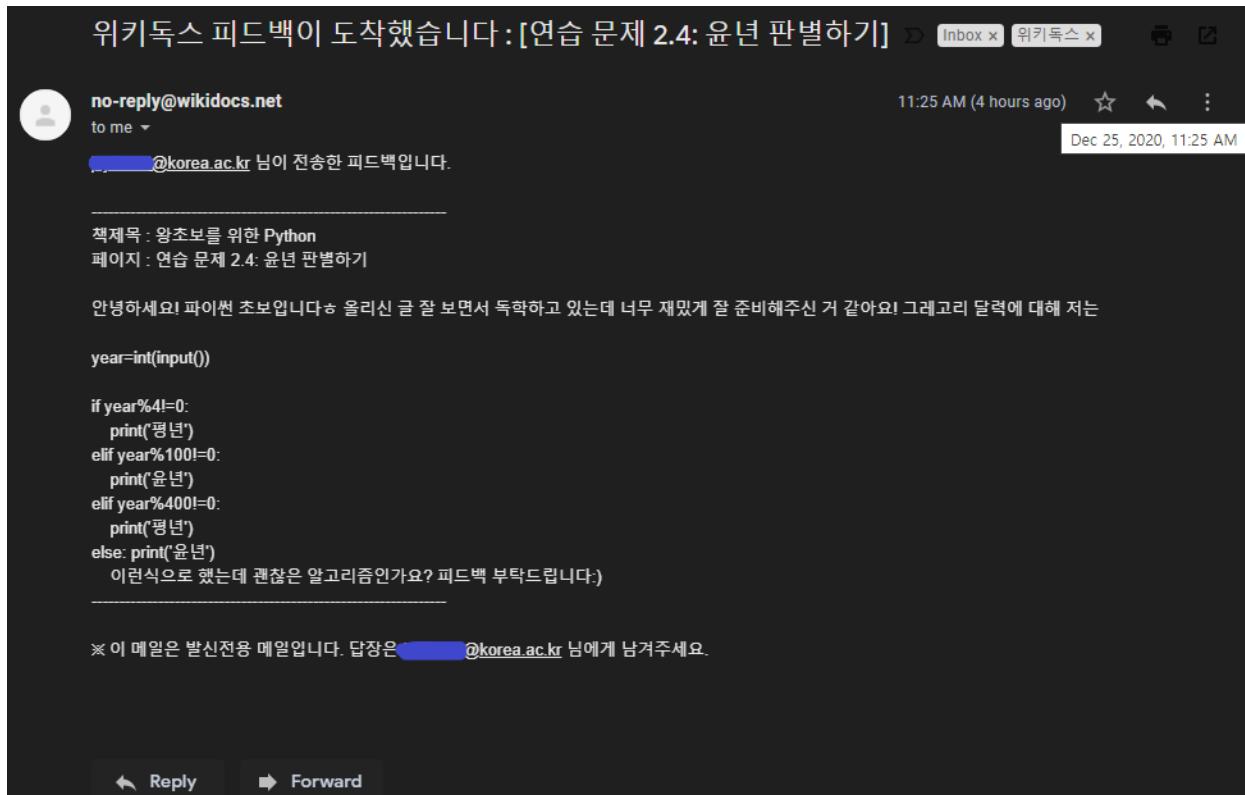
- 테스팅
- 프로그램 실행 시간 측정하기



9.1 테스팅

오늘은 크리스마스입니다. 메리 크리스마스 ~

독자께서 [윤년 판별하기 연습 문제](#)를 푼 것을 봐달라고 메일로 보내주셔서 한번 살펴보려고 합니다.



저는 [if 문을 중첩해서 풀었었는데](#), 이 코드는 중첩 없이 평평하게 되어 있어서 보기 좋네요. 시각적으로 보기 좋은 코드가 실제로도 좋을 가능성이 높습니다.

그렇지만 코드를 눈으로만 봐서는 결과가 제대로 나올지 확신하기 힘들므로, 실제 숫자를 넣어서 테스트 해보고, 그다음에는 파이썬 유닛테스트 (unittest) 모듈을 사용해 테스트를 자동화하려고 합니다.

주어진 연도가 윤년인지 아닌지를 반환하는 함수

우선 테스트하기 편하게 코드를 약간 바꿔볼게요. 원래는 `input()` 함수로 입력을 받아서 `print()` 문으로 결과를 출력하게 되어 있었는데, `is_leap_year()` 함수로 만들었습니다. 이 함수는 `year` 매개 변수가 윤년에 해당하면 `True`를, 그렇지 않으면 `False`를 반환합니다.

```
# 파일명: LeapYear.py

def is_leap_year(year):
    if year % 4 != 0:
        return False
    elif year % 100 != 0:
        return True
    elif year % 400 != 0:
        return False
    else:
        return True
```

IDLE에서 위와 같이 작성해 `LeapYear.py`로 저장하고 **F5** 키를 눌러 실행합니다.

```
>>>
= RESTART: C:\Users\ychoi\OneDrive\문서\GitHub\wikidocs-chobo-python\ch09\LeapYear
.py
```

`LeapYear.py`가 다시 시작되었다는 메시지가 떴습니다. 즉, `is_leap_year()` 함수가 정의되었겠죠? 팔호 없이 함수 이름만 넣어서 확인해보겠습니다.

```
>>> is_leap_year
<function is_leap_year at 0x000002007E874DC0>
```

네, `is_leap_year`라는 함수가 메모리 주소 `0x000002007E874DC0`에 있다고 합니다. 메모리 주소는 집 주소와 비슷합니다. 여러분이 온라인 쇼핑에서 주문한 물건을 배송 받으려면 집 주소를 알려줘야 하는 것처럼, 컴퓨터 메모리에 저장해둔 함수를 사용하려면 메모리 주소가 필요한 것이죠. 하지만 우리는 메모리 주소를 몰라도 됩니다.

그럼 `is_leap_year()` 함수를 사용해 보겠습니다. 올해는 2020년이니까 2020을 넣어보죠.

```
>>> is_leap_year(2020)
True
```

2020년은 윤년이므로 `True`가 나왔습니다.

이번에는 2077로 테스트해봅시다. 2077년은 평년이므로 `False`가 나와야겠죠?

```
>>> is_leap_year(2077)
False
```

네, 좋습니다.

주어진 연도가 윤년인지의 여부를 반환하는 `is_leap_year()` 함수를 작성하고 수작업으로 간단히 테스트해보았습니다.

유닛 테스트 작성과 실행

이제, 위에서 수작업으로 수행했던 테스트 과정을 자동화하는 코드를 작성해보겠습니다. 파이썬 언어에 기본으로 포함된 `unittest` 모듈을 이용할 거구요, 실제 코드와 테스트 코드를 같은 파일에 넣어도 되겠지만 그렇게 하지 않고 `test_LeapYear`라는 이름의 파일을 따로 작성하겠습니다.

```
# 파일명: test_LeapYear.py

1 import LeapYear
2 import unittest
3
4 class TestLeapYear(unittest.TestCase):
5     def test_2020(self):
6         r = LeapYear.is_leap_year(2020)
7         self.assertEqual(r, True)
8
9 if __name__ == '__main__':
10     unittest.main()
11
```

코드 설명:

- 1 행: 위에서 작성한 `LeapYear` 모듈을 임포트
- 2 행: `unittest` 모듈을 임포트
- 4~7 행: `LeapYear` 모듈을 테스트하는 `TestLeapYear` 클래스 정의
 - 5~7 행: `is_leap_year()` 함수를 테스트하는 메서드
 - * 6 행: `is_leap_year(2020)`을 실행한 결과를 `r`이라는 이름으로 저장합니다. `r`의 값은 2020년이 윤년인지 아닌지 나타내는 불린 값 (`True` 또는 `False`)이 됩니다.
 - * 7 행: 'r 값이 `True`여야 올바른 결과이다'라는 의도를 코드로 표현한 것입니다.
- 9~10 행: `test_LeapYear`를 실행하면 테스트를 실행

IDLE에서 위와 같은 테스트 코드를 작성해 `test_LeapYear.py`로 저장한 다음, **F5** 키를 눌러 실행하면 IDLE Shell 창에 다음과 같은 결과가 나옵니다.

```
>>>
= RESTART: C:\Users\ychoi\OneDrive\문서\GitHub\wikidocs-chobo-python\ch09\
    test_LeapYear.py
.

-----
Ran 1 test in 0.011s

OK
```

테스트를 한 개 수행했고, 모든 테스트를 수행한 결과가 좋다는 뜻으로 `OK`라고 출력되었습니다.

테스트 추가

2020년에 대해서 테스트를 작성해봤으니, 2077년에 대해서도 테스트해보겠습니다. 앞에서 작성한 `TestLeapYear` 클래스에 `test_2077()` 메서드를 추가합니다.

```
# 파일명: test_LeapYear.py

(앞 부분 생략)
8
9     def test_2077(self):
10         r = LeapYear.is_leap_year(2077)
11         self.assertEqual(r, False)
12
13 if __name__ == '__main__':
14     unittest.main()
```

이 메서드를 앞에서 작성한 `test_2020()` 메서드와 비교해보면 한 가지 다른 점이 있습니다. 찾으셨나요? 맞습니다. 11행의 두 번째 인자가 `False`로 되어 있죠. 2077년은 윤년이 아니기 때문에, `is_leap_year(2077)`의 실행 결과는 `False`여야 한다는 것을 이렇게 나타냈습니다.

그럼, 테스트를 다시 실행해보겠습니다.

```
>>>
= RESTART: C:\Users\ychoi\OneDrive\문서\GitHub\wikidocs-chobo-python\ch09\
    test_LeapYear.py
.

-----
Ran 2 tests in 0.012s

OK
```

이번에는 테스트 두 개가 실행되었고, 둘 다 통과해서 OK가 나왔습니다.

모든 분기를 테스트했나?

지금까지 `LeapYear.py` 코드를 테스트해봤습니다. 그런데 테스트가 이것으로 충분할까요, 아니면 테스트를 좀 더 해봐야 할까요?

우리는 코드를 갖고 있으므로, `if` 문을 사용한 분기가 어떤 구조인지 알고 있습니다. 경우의 수가 네 가지 있는데, 각각 한 번씩은 테스트해보는 것이 좋을 것 같습니다. 그렇다면 앞에서 테스트한 분기는 어느 것이고, 테스트하지 않은 분기는 어느 것일까요?

테스트 대상인 `LeapYear.py` 코드를 다시 보면서 생각해보겠습니다.

```
# 파일명: LeapYear.py

def is_leap_year(year):
    if year % 4 != 0:          # (1) 2077
        return False
    elif year % 100 != 0:       # (2) 2020
        return True
    elif year % 400 != 0:       # (3) ?
        return False
    else:                      # (4) ?
        return True
```

위 코드에서 (1) '연도를 4로 나누어 떨어지지 않으면 윤년이 아니다'라는 논리를 작성했는데, 이를 테스트 코드에서는 2077을 가지고 테스트했었죠? 다음으로, (2) '연도를 4로 나누어 떨어지되 100으로 나누어 떨어지지 않으면 윤년이다'라는 논리를 2020으로 테스트했습니다. (3) 번과 (4) 번 논리는 테스트하지 않았는데, 이것들이 올바로 작성되었는지도 추가로 테스트할 필요가 있겠네요.

숙제

그러면 (3) 번과 (4) 번을 테스트하려면 몇 년을 가지고 테스트해야 할까요? 400년 주기가 중요하므로 2000년과 1900년을 살펴보면 될 것 같습니다. 한번 작성해보세요.

잘 작성했다면 결과가 다음과 같이 나올 거예요.

```
= RESTART: C:\Users\ychoi\OneDrive\문서\GitHub\wikidocs-chobo-python\ch09\
  test_LeapYear.py
  ...
-----
Ran 4 tests in 0.013s
```

OK

더 읽을 거리

- 실용 파이썬 프로그래밍 - 8.1 테스팅: <https://wikidocs.net/84431>

9.1.1 연습 문제: 숫자 읽기 (0~100)

문제

0 이상 100 이하의 정수를 매개변수로 입력 받아 그 숫자에 해당하는 한글 문자열을 반환하는 함수 `korean_number()`를 포함하는 `korean_0_to_100.py`를 작성하세요. 아래 테스트 (`test_korean_number.py`)를 이용하되, 테스트 코드를 추가해도 됩니다.

```
from korean_0_to_100 import korean_number
import unittest

class TestKoreanNumber(unittest.TestCase):
    def test_0(self):
        self.assertEqual(korean_number(0), '영')

    def test_1(self):
        self.assertEqual(korean_number(1), '일')

    def test_2(self):
        self.assertEqual(korean_number(2), '이')

if __name__ == '__main__':
    unittest.main()
```

답

- `ch09/test_korean_number.py`
- `ch09/korean_0_to_100.py`

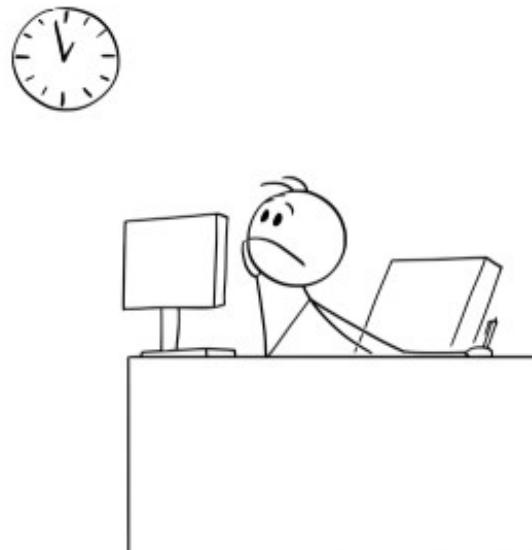
영상

- <https://youtu.be/m2q1uR7GsXU>

9.2 프로그램 실행 시간 측정하기

같은 목적으로 작성한 프로그램이라 하더라도, 그 논리를 어떻게 작성했는지에 따라 계산이 오래 걸리기도 하고, 상대적으로 일찍 끝나기도 합니다. 그렇다면, 프로그램을 실행하는 데 걸리는 시간을 어떻게 측정할 수 있을까요?

시계를 보면서 프로그램의 시작 시각과 종료 시각을 확인해서 걸린 시간을 계산하면 될까요?



time.process_time()

이렇게 프로그램을 실행하는 데 실제로 걸린 시간 (벽시계에 비유해 [wall time](#)이라고 합니다) 을 알고 싶을 때도 있겠지만, 컴퓨터의 처리 시간 ([process time](#)) 을 계산하는 것이 나을 때도 있습니다. 파이썬 프로그램의 처리 시간을 알고 싶을 때 [time](#) 모듈의 [process_time](#)으로 시간을 측정할 수 있습니다.

[4장 소수 구하기 연습 문제](#)의 첫 번째 풀이 ([prime.py](#)) 의 실행 시간을 측정하려면 다음과 같이 할 수 있습니다. 윈도우 명령 프롬프트의 작업 디렉터리를 [ch09](#)에서 [ch04](#)로 바꾼 뒤 파이썬 인터프리터를 실행했습니다.

```
wikidocs-chobo-python\ch09> cd ..  
wikidocs-chobo-python> cd ch04  
  
wikidocs-chobo-python\ch04> python  
Python 3.9.4 (tags/v3.9.4:1f2e308, Apr 6 2021, 13:40:21) [MSC v.1928 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import prime  
>>> from time import process_time  
>>> start = process_time()  
>>> prime.prime(2 ** 8)          # 256 이하의 소수  
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,  
 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157,  
 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239,  
 241, 251]  
>>> end = process_time()  
>>> end - start  
0.03125  
>>> exit()  
wikidocs-chobo-python\ch09>
```

처리 시간 (`end - start`)이 0.03125초 걸렸다고 나왔습니다.

sys.path.append()

이번에는 두 번째 폴더 (`prime2.py`)의 실행 시간을 구해보겠습니다. 위에서는 윈도우 명령 프롬프트에서 직접 `ch04` 폴더로 이동해서 파이썬을 실행했지만, 아래와 같이 `ch09`에서 파이썬을 실행해 `sys.path.append()`로 `ch04`를 패키지 경로에 추가하고 모듈을 임포트할 수도 있습니다.

```
wikidocs-chobo-python\ch04>  
wikidocs-chobo-python\ch04> cd ../ch09  
  
wikidocs-chobo-python\ch09> python  
Python 3.9.4 (tags/v3.9.4:1f2e308, Apr 6 2021, 13:40:21) [MSC v.1928 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import sys  
>>> sys.path.append('../ch04')  
>>> import prime2  
>>> from time import process_time  
>>> start = process_time()  
>>> prime2.prime(2 ** 8)  
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,  
 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157,  
 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239,  
 241, 251]  
>>> end = process_time()
```

```
>>> end - start  
0.0  
>>>
```

이번에는 처리 시간이 0.0으로, 같은 문제를 푸는 데 시간이 훨씬 적게 걸렸습니다.

importlib.import_module()

그 밖에 독자분들이 알려주신 풀이도 있어서 이름이 prime으로 시작하는 파일이 여러 개 있습니다. 이 파일들의 실행 시간을 모두 비교하는 프로그램을 작성했는데, 모듈명을 제가 직접 지정하지 않고 glob을 이용해 알아낸 후 반복문으로 하나씩 실행했습니다. 이때 모듈명을 나타내는 문자열을 가지고 임포트하기 위해 importlib의 import_module() 함수를 사용했습니다. 그리고 프로그램의 성능을 더 잘 비교하기 위해 N을 아까보다 더 큰 수로 지정했습니다.

[코드: ch09/prime_process_time.py]

```
import sys  
import os  
import importlib  
from glob import glob  
from time import process_time  
  
N = 2 ** 12          # 4096  
# N = 2 ** 13        # 8192  
  
# 현재 장은 9장인데 소수 구하기 코드는 4장에 있어, 패키지 경로를 추가하고 작업 디렉터리도 변경  
sys.path.append('../ch04')  
os.chdir('../ch04')  
  
for pth in glob('prime*'): # prime으로 시작하는 파일명 각각에 대해...  
    name = os.path.splitext(pth)[0] # 모듈명(파일명에서 확장자 앞까지) 구하기  
    print(f'\nRunning {pth} ...')  
    mod = importlib.import_module(name) # 모듈 임포트  
  
    # 실행 시간 측정  
    start = process_time()  
    mod.prime(N)  
    end = process_time()  
    print('elapsed:', end - start)
```

실행 결과 ($N = 2^{12}$ 일 때)

```
wikidocs-chobo-python\ch09>prime_process_time.py

Running prime.py ...
[2, 3, 5, (생략), 4079, 4091, 4093]
elapsed: 57.15625

Running prime2.py ...
[2, 3, 5, (생략), 4079, 4091, 4093]
elapsed: 0.0

Running prime_aaaa.py ...
Prime Number List Of 4096
[2, 3, 5, (생략), 4079, 4091, 4093]
elapsed: 0.015625

Running prime_fate.py ...
[2, 3, 5, (생략), 4079, 4091, 4093]
elapsed: 0.015625

Running prime_kim.py ...
[2, 3, 5, (생략), 4079, 4091, 4093]
elapsed: 0.671875
```

실행 결과 ($N = 2^{13}$ 일 때)

다음은 N 값을 2^{13} 으로 바꿔서 실행해본 결과입니다.

```
wikidocs-chobo-python\ch09>prime_process_time.py

Running prime.py ...
[2, 3, 5, (생략), 8171, 8179, 8191]
elapsed: 444.125

Running prime2.py ...
[2, 3, 5, (생략), 8171, 8179, 8191]
elapsed: 0.015625

Running prime_aaaa.py ...
Prime Number List Of 8192
[2, 3, 5, (생략), 8171, 8179, 8191]
elapsed: 0.09375

Running prime_fate.py ...
[2, 3, 5, (생략), 8171, 8179, 8191]
elapsed: 0.046875

Running prime_kim.py ...
[2, 3, 5, (생략), 8171, 8179, 8191]
elapsed: 4.078125
```

A. 부록

여기까지 읽으신 여러분은 이제 '왕초보'는 아니고 '초보'입니다!?

독자 여러분이 특히 어려워 하셨던 문제 몇 개는 파이썬에 익숙해진 후에 풀면 좋을 것 같아서 이곳으로 옮겼습니다. 그 외에 파이썬 기초 문법은 아니지만 익혀두면 유용한 것들을 모았습니다.

- [함수의 재귀](#)
- [스케줄링 알고리듬](#)
- [진법 변환과 비트 연산](#)
- [파이썬으로 PDF 파일 합치기](#)
- [matplotlib으로 하트 그리기](#)
- [윈도우에서 파이썬 활용 팁](#)



A.1 함수의 재귀

이번에 배울 것은 새로운 파이썬 문법은 아니구요, 프로그램을 짜는 테크닉 중의 한 가지인데 조금 머리가 아플 수도 있는 내용이랍니다. 하지만 최대한 쉽게 알려드릴 테니까 너무 걱정 마시고 함께 알아보도록 해요. (3장에 있던 것을 부록으로 옮겼습니다.)

함수가 자기 자신을 호출하는 ‘재귀 (recursion)’ 혹은 ‘순환’입니다.

갑자기 어려운 것을 배우면 머리가 많이 아프실 테니까 먼저 준비운동을 하는 것이 좋겠군요. 다음의 예제를 봐주세요. 먼저 주어진 두 수를 합하는 함수를 만들어보겠습니다.

```
>>> def hap(a, b):
...     print(a + b)
...
```

제대로 만들었는지 확인을 해보세요. 이 함수를 어떻게 쓰는지 아시겠죠? 확인해보셨으면 두 수를 곱하는 함수도 만들어보세요.

```
>>> def gop(a, b):
...     print(a * b)
...
```

그럼 이번엔 두 수를 합해보고 곱해보고, 두 가지 일을 다하는 함수도 만들어 보겠습니다.

```
>>> def hap_gop(a, b):
...     hap(a, b)
...     gop(a, b)
...
```

이 함수는 자기에게 맡겨진 일을 직접 수행하지 않고 다른 함수들에게 시켜버리죠? 다시 말하면, 이 함수는 `hap()` 함수와 `gop()` 함수를 호출했다고 할 수 있습니다. 한번 테스트해보세요. 재미 있으신가요?

이젠 본론으로 들어가겠습니다.

```
def countdown(n):
```

```
if n == 0:  
    print("Blastoff!")  
else:  
    print(n)  
    countdown(n-1)
```

- 출처: [How to Think Like a Computer Scientist](#)

이 함수는 무슨 일을 하는 함수일까요? 함수 이름을 보면 추측을 할 수도 있는데... 잘 모르시겠나요? 그렇다면 몇 줄 안되니까 직접 쳐보시지요.

제대로 만드셨으면 아래와 같이 사용을 해보시구요.

```
>>> countdown(3)  
3  
2  
1  
Blastoff!
```

이 함수는 매개변수로 받은 수부터 카운트다운을 하다가 0 까지 오면 꽝! 하는 일을 한답니다. '그런 거라면 for 문이나 while 문과 똑같잖아' 하고 생각하는 분도 계시겠지만 코드를 가만히 보시면 차이점을 발견하실 거예요.

이 함수는 if, else 구조로서 n 이 0 인지 아닌지 따라서 다른 일을 하도록 되어 있습니다. 이 if 문에서 검사하는 n 은 첫째줄에서 함수를 정의할 때 매개변수로 정해준 n 과 같은 녀석입니다. 그러니까 매개변수에 3 을 넣어서 함수를 호출하면, 즉 `countdown(3)` 이라고 쓰면 함수 본체의 n 값으로 3 이 들어가는 것지요.

n 값으로 3 이 들어오면 함수 내부에서는 어떤 일이 벌어질까요?

먼저 `if n == 0:` 에서 n 값이 0 과 같은지 비교합니다. n 이 3 이므로 n 의 값과 0 은 같지 않습니다. 그러면 `else:` 이후의 명령을 수행하겠지요.

다음 줄에 있는 `print(n)` 은 n 값을 출력하라는 명령이니까 화면에 3 을 출력해줍니다.

그 다음엔 `countdown(n-1)` 이라고 되어있지요. `countdown` 이라는 함수를 호출하면서 n-1 값을 매개변수로 넣어주라는 뜻입니다. 결국 `countdown(2)` 와 같이 함수를 호출하게 되지요.

그런데 여기서 뭔가 이상하다는 것을 눈치채셨을 겁니다.

`countdown` 이라는 함수에서 `countdown` 을 호출한다?

예, 그것이 바로 오늘의 주인공, 재귀적 호출입니다. 함수가 자기 자신을 호출하는 것이죠.

머리가 복잡해지기 시작하신다면 생각을 잠시 덮으시고 `countdown(2)` 라고 호출하면 어떤 일이 일어날지부터 함께 따져보죠.

먼저 `if` 문에서 `n` 값이 0과 같은지 검사를 합니다. 검사 결과가 거짓이므로 `else:` 이후의 명령을 수행하겠죠. `print(n)`에서 `n` 값인 2를 화면에 출력해주고 그다음 줄에서 `countdown(1)`을 호출합니다.

오호~.

뭔가 감이 잡히시나요? 처음에 `countdown` 함수를 호출하면서 매개변수 값으로 3을 넣어줬는데 함수가 자기자신을 호출하면서 매개변수가 점점 작아지죠? 그렇다면 `countdown(1)`은 다시 `countdown(0)`을 호출하게 되는 것이 틀림없습니다.

그러면 `countdown(0)`에서는 무슨 일을 할까요? `n == 0`이면 `print("Blastoff!")` 하라고 되어있으니 화면에 그렇게 출력해줍니다. ‘뻥이야~!’

그렇게해서 `countdown` 함수가 끝이 납니다. 지금까지 본 것이 모두 `countdown(3)`을 호출했을 때 일어나는 일들입니다.

이 일들을 하는 동안 `countdown` 함수는 네 번이나 호출되었지요. `countdown(3), countdown(2), countdown(1), countdown(0)`. 맞나요?

어떠세요? 이해할 만한가요? 좀 어렵긴 하지만 알고 나면 꽤 재미가 있지요.

사실 이 예제에서는 `for` 문을 이용해도 얼마든지 할 수 있는 일을 예로 들었지만 어떤 경우에는 재귀적 호출을 잘 사용하면 복잡한 프로그램을 아주 쉽게 만들 수도 있답니다.

하지만, 함수를 여러 번 호출해서 일을 처리하기 때문에 사람이 편한 만큼 컴퓨터에게는 힘이 더 드는 방법이기도 하답니다.

이 내용은 지금 꼭 이해하지 못해도 상관없습니다. 프로그래밍 공부를 계속하시다보면 다시 배우게 될테니까요. 예전에 배운 내용을 확실히 몰라서 이 내용을 이해하지 못하셨다면 복습을 좀 더하신 후에 다시 보시는 것이 좋겠네요.



A.1.1 연습 문제: 각 자리 숫자의 합을 구하는 재귀 함수

문제¹

어떤 수 (number)의 각 자리 숫자 (digit)의 합을 계산하는 `sumOfDigits()`라는 재귀 함수를 작성하자. 입력한 수를 읽어 `sumOfDigits()` 함수를 호출하며, 이 함수는 합산할 숫자가 남지 않을 때까지 자신을 호출해, 최종적인 합을 사용자에게 표시한다.

`sumOfDigits()`는 다음과 같은 원리로 작동한다.(주의: 실제 코드가 아님)

```
sumOfDigits(6452) = 2 + sumOfDigits(645)
sumOfDigits(645) = 5 + sumOfDigits(64)
...
sumOfDigits(6) = 6
```

예 1

입력:

```
47253
```

출력:

```
21
```

예 2

입력:

¹edX.org 의 C Programming: Modular Programming and Memory Management에 실린 문제입니다.

643

출력:

13

풀이

아래 코드는 문제에서 요구한 대로 재귀 함수를 정의했습니다.

- 코드: [ch10/sumOfDigits_recursive.py](#)
- [흐름도](#)

다음 코드는 재귀적이지 않은 함수를 정의해서 풀었습니다.

- [ch10/sumOfDigits_non-recursive_div-mod.py](#)

A.1.2 연습 문제: 복리를 계산하는 재귀 함수

[복리 계산 문제](#)를 재귀적으로 풀어봅시다.

문제

문제 1

문제를 단순화하기 위해 복리 계산 공식을 조금 바꿔 볼게요. 원래 공식에는 복리 횟수가 있는데, 복리 횟수가 1이라고 가정하면 공식을 다음과 같이 간단히 나타낼 수 있습니다.

$$P' = P(1 + r)^t$$

P : 원금

P' : 원리금

r : 이율

t : 기간

그런데 이번 문제는 위 공식을 그대로 사용하지 말고 재귀적으로 풀어야 합니다. 결괏값의 소수점 이하는 그대로 두시면 됩니다.

예 1

연이율 5.8% 인 연복리 상품에 360 만 원을 2년간 예치했을 때 만기 수령액:

```
>>> compound_interest_amount_withoutN(3600000, 0.058, 2)  
4029710.400000004
```

예 2

천만 원을 연이율 5% 월복리 예금에 1년 넣었을 때 만기 수령액:

```
>>> 0.05 / 12  
0.00416666666666667  
>>> compound_interest_amount_withoutN(10000000, _, 12)  
10511618.978817329
```

문제 2 (심화)

위 문제를 풀었다면 아래 공식으로도 풀어보세요.

$$P' = P\left(1 + \frac{r}{n}\right)^{nt}$$

P : 원금

P' : 원리금

r : 이율

n : 복리 횟수

t : 기간

이 문제를 풀 수 있다면 엄청 잘 하시는 것으로 인정! 아, 재귀적으로 풀었을 때만요.

예 1

```
>>> compound_interest_amount(1500000, 0.043, 6, 4)  
1938836.8221341053
```

예 2

```
>>> compound_interest_amount(1500000, 0.043, 6, 1/2)  
1921236.0840000005
```

예 3

함수를 만들었다면 마지막 인자로 1을 전달해서 1 번 문제도 풀 수 있어요.

```
>>> compound_interest_amount(3600000, 0.058, 2, 1)  
4029710.400000004
```

풀이

- 코드: [ch10/compoundInterest_recursive.py](#)
- 구글 스프레드시트
- [흐름도](#)

참고

- [복리 계산법, 탱스타](#)
- [예적금 단리 복리 차이점 알아보기, 치킨요정의 경제공부방](#)

A.2 연습 문제: 여러 대의 컴퓨터에 연산을 분배하기

4 장에 있던 연습 문제인데 좀 어려워서 뒤로 옮겼습니다. 한번 풀어보세요.

문제

컴퓨터가 몇 대 있고 연산해야 할 프로그램도 몇 개 있습니다. 프로그램들을 컴퓨터에 가장 적절하게 분배하는 프로그램을 작성하세요.

예) 컴퓨터는 2 대가 있고, 프로그램의 수행시간은 각 3 분, 5 분, 2 분이라면, 컴퓨터 하나는 3 분, 2 분짜리 프로그램을 수행하고 다른 컴퓨터는 5 분짜리 프로그램을 수행하면 됩니다.

```
입력
computer : 2
program : 3, 5, 2
출력
computer1 : 5
computer2 : 3, 2
```

어떠세요? 좀 어렵죠? 저도 이 문제를 보고 문제가 무슨 뜻인지 몰라서 한참 헤맸답니다. 그러나 총명하신 여러분은 금방 이해하셨으리라 생각합니다.

이제 문제를 풀어볼 텐데요, 아래의 해설을 보시기 전에 문제를 다른 곳에 옮겨놓고 잠시 생각을 해보시기 바랍니다. 가능하면 직접 풀어보는 것이 좋으니까요.

컴퓨터 여러 대가 프로그램을 나눠서 수행한다면, 각각의 컴퓨터에게 같은 양의 일을 줘서 같이 끝내는 것 이 좋겠죠? 예에서는 총 10 분 동안 할 일을 컴퓨터 두 대에게 나눠주는 거니까 각각 5 분씩 일을 시키면 되는 거구요.

만약, 수행할 프로그램 중에 평균보다 더 오래 걸리는 것이 있다면, 그러니까 프로그램이 7 분, 3 분 두 개 가 주어지면 어떻게 나눠주는 것이 좋을까요? 할 수 없이 한 대는 7 분 동안, 다른 한 대는 3 분 동안 일을 해야겠죠? 7 분 짜리를 5 분, 2 분으로 쪼개서 넘겨주라고요? 시로 ~.

해법

문제를 두 가지 방법으로 풀어 보았습니다. 첫 번째 방법은 `sol1()` 함수, 두 번째 방법은 `sol2()` 함수로 구현했습니다. 먼저 두 함수에 공통으로 들어간 코드를 설명드릴게요.

공통적인 부분

함수 입출력 형식

예제에서는 단순하게 리스트와 변수를 입력받고, 리스트를 출력하도록 처리했습니다. 그렇게 하는 것이 간편하고 문제에 집중하는 데 도움이 되는 것 같아서요.

입력:

- 첫 번째 매개변수: 리스트 (예: [3, 5, 2])
- 두 번째 매개변수: 정수 (예: 2)

출력:

- 리스트 (예: [[5], [3, 2]])

초기화

두 함수의 첫 부분에는 다음 코드가 공통으로 들어갑니다.

```
_inlist = copy.deepcopy(inlist)
outlist = []
sumout = []

for x in range(coms):
    outlist.append([])
    sumout.append(0)
```

첫째 줄에서는 매개변수로 받은 `inlist`를 `_inlist`로 깊은 복사를 했습니다.

`outlist`가 바로 컴퓨터들의 리스트이고, `sumout`은 컴퓨터마다 수행할 프로그램의 수행 시간 합계를 갖는 리스트입니다.

`coms`는 함수의 매개변수로 받은 컴퓨터 대수입니다. `coms` 값이 3이라면 위의 `for` 문을 수행한 후에 `outlist`는 [[], [], []]와 같이 되고, `sumout`은 [0, 0, 0]이 됩니다. 나중에 함수가 모두

실행되고 나면 `outlist`에는 결괏값이 [[22], [15, 3, 2], [13, 6]]와 같이 담기고, 그때 `sumout`은 [22, 20, 19]와 같이 되어있도록 할 생각이지요.

해법 1

문제 해결을 위한 첫 번째 해법의 논리는 다음과 같습니다.

프로그램들을 전부 수행하는 데 걸리는 총수행시간을 구한다.
총수행시간을 컴퓨터 대수로 나누어 각각의 컴퓨터의 평균수행시간을 구한다.
만약 평균수행시간보다 더 오래 걸리는 프로그램이 있으면:
그냥 컴퓨터에게 준다.
평균 수행시간보다 짧게 걸리는 것들은:
평균수행시간만큼 모아서 컴퓨터에게 준다.

코드는 다음과 같습니다.

```
def sol1(inlist, coms):
    ...
    # 프로그램들을 전부 수행하는 데 걸리는 총수행시간을 구한다.
    total_time = sum(_inlist)

    # 총수행시간을 컴퓨터 대수로 나눠 각 컴퓨터의 평균수행시간을 구한다.
    avg_time = total_time / coms

    for j in range(coms):
        # 평균수행시간보다 오래 걸리는 프로그램이 있으면 그냥 컴퓨터에게 준다.
        if True in [k >= avg_time for k in _inlist]:
            for m in _inlist:
                if m >= avg_time:
                    outlist[j].append(m)
                    _inlist.remove(m)
                    sumout[j] += m
                    break

        # 평균 수행시간보다 짧게 걸리는 것들은 모아서 컴퓨터에게 준다.
        else:
            for n in range(len(_inlist)):
                if _inlist[n] == 0:
                    continue
                v = _inlist[n]
                outlist[j].append(v)
                _inlist[n] = 0
                sumout[j] += v
                if sumout[j] >= avg_time:
                    break

    return outlist
```

처음에는 이런 식으로 문제를 풀었는데, 나중에 다시 보니까 복잡하면서도 비효율적이라는 생각이 들더군요.

해법 2

그래서 몇 날 며칠을 고민하다보니 기발한 아이디어가 떠올랐습니다. 새로운 해결방법의 핵심은 ‘가장 가벼운 바구니에 빵을 담는다’라는 것입니다. ‘컴퓨터’와 ‘수행할 프로그램’ 대신 ‘바구니’와 ‘빵’이라고 생각한 것이죠. 그렇게 되면, 컴퓨터에게 프로그램을 분배하는 것은 바구니에 빵을 하나씩 담는 것과 흡사하지요.

```
바구니를 준비한다.  
바구니에 담을 빵들을 크기순으로 정렬한다.  
빵을 모두 바구니에 담을 때까지:  
    가장 가벼운 바구니에 가장 큰 빵을 담는다.  
결과를 돌려준다.
```

이 논리를 프로그램으로 구현한 것이 아래의 예제입니다. 대화식으로 작업하는 것보다 텍스트 에디터를 사용해서 프로그램 파일로 작성하는 것이 좋겠죠?

```
def sol2(inlist, coms):  
    ...  
  
    _inlist.sort(reverse=True)  
  
    # 수행 할 작업을 빵(bread)에, 컴퓨터를 바구니(basket)에 비유  
    for bread in _inlist:  
        lowbasket = sumout.index(min(sumout)) # 가장 가벼운 바구니에  
        outlist[lowbasket].append(bread) # 빵을 담는다  
        sumout[lowbasket] += bread  
  
    return outlist
```

정말 간단하죠? 예전에 소개해드렸던 풀이를 보셨던 분이라면 놀라실 거예요. 저 스스로도 감탄했거든요.

프로그램을 전체적으로 보면 `sol2()`라는 함수를 하나 정의해두고, 뒤에 가서 그 함수를 호출하는 형태입니다. 그리고, `sol2()` 함수는 `inlist`와 `coms`라는 두 개의 매개변수를 받아서, `outlist`를 결과로 돌려줍니다.

이제 함수의 내부를 살펴봅시다. “바구니에 담을 빵들을 크기순으로 정렬한다.”

```
_inlist.sort(reverse=True)
```

`_inlist`를 `sort()` 메서드를 써서 정렬하되, `reverse` 옵션에 `True`를 지정함으로써 정렬이 역순(내림차순)으로 이루어지게 합니다.¹ 다음과 같이 두 줄로 쓴 것과 같은 결과를 얻을 수 있습니다.

```
_inlist.sort()  
_inlist.reverse()
```

이제 각각의 빵을 바구니로 분배하겠습니다.

```
for bread in _inlist:  
    lowbasket = sumout.index(min(sumout)) # 가장 가벼운 바구니에  
    outlist[lowbasket].append(bread) # 빵을 담는다  
    sumout[lowbasket] += bread
```

가장 가벼운 바구니를 구하기 위해 `min(sumout)`을 했고, 그 바구니의 인덱스를 구해서 `lowbasket`이라고 했습니다. 각각의 바구니는 `outlist`와 `sumout`에서 같은 인덱스를 쓴다는 점을 이용하기 위해서이지요. 예를 들어서, 만약 `outlist` 전체가 `[[22], [15], [13, 6]]`인 상태라면 `sumout`은 `[22, 15, 19]`가 되어있을 테고요, 현재 가장 가벼운 바구니는 15가 들어있는 바구니입니다.

```
>>> outlist = [[22], [15], [13, 6]]  
>>> sumout = [22, 15, 19]  
>>> min(sumout)  
15
```

이것의 인덱스를 알아보면 1이라는 걸 알 수 있지요.

```
>>> sumout.index(15)  
1
```

이제 구해진 인덱스를 `lowbasket`이라고 하고, `outlist`와 `sumout`에서 인덱스 1인 바구니는 다음과 같습니다.

```
>>> outlist[1]  
[15]  
>>> sumout[1]  
15
```

이런 방법으로 두 리스트에서 같은 바구니를 가리키는 부분을 찾아서 바구니에 빵을 담을 때, 그 바구니가 얼마나 찬지도 함께 체크해두는 겁니다. `_inlist`의 모든 원소에 대해 이 일을 반복했으면, 즉 모든 빵을 바구니에 나눠담았으면, 구해진 `outlist`를 함수의 결괏값으로 돌려줍니다.

¹Lilly 님께서 알려주셨습니다.

```
return outlist
```

끝이에요 ~. 이제 프로그램을 한번 돌려보세요. 아주 잘 ~ 될거예요 ~²

보충 설명

아래는 지성님께서 설명해주신 내용입니다.

이것은 프로세스 스케줄링 방식 중에 LPT(Longest Processing Time) 스케줄링에 해당합니다. 가장 긴 시간을 요하는 Job 부터 차례대로 가장 적은 업무를 맡은 Line 에 배분하는 방식이지요 ~ 실제로 이 방식의 경우 100% 최적해를 찾는다고 말할 수는 없지만, 이 방식을 사용하면 Parallel Line(컴퓨터가 2 대) 에서 최적해보다 33% 이상 나쁘지 않고, Line 의 수가 증가할수록 최적해에 가까운 결괏값을 도출하게 된답니다! Line 의 수가 많아지고 Job 의 수가 많아질수록 최적해를 구하기 위하여 고려해야 하는 결과의 가짓수가 매우 많아지기 때문에, 이런 단순 스케줄링의 경우에 사용하기 매우 경제적인 방법이죠!

다음은 예시입니다 ~!

| | | | | | | | | | | | |
|------|---|---|---|---|---|---|---|---|---|----|----|
| Jobs | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Pt | 9 | 9 | 8 | 8 | 7 | 7 | 6 | 6 | 5 | 5 | 5 |

Pt: Processing time, # of Machines = 5

최적해:

| | | | | | | | | | | | | | | | |
|--------|---|---|---|---|-------|-------|---|---|--------|----|----|----|--------|----|----|
| Hour | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Line 1 | [| | | | | Job 1 | | | |] | [| | Job 8 | |] |
| Line 2 | [| | | | | Job 2 | | | |] | [| | Job 7 | |] |
| Line 3 | [| | | | | Job 3 | | | |] | [| | Job 6 | |] |
| Line 4 | [| | | | | Job 4 | | | |] | [| | Job 5 | |] |
| Line 5 | [| | | | Job 9 | |] | [| Job 10 | |] | [| Job 11 | |] |

LPT:

| | | | | | | | | | | | | | | | | | | | |
|--------|---|---|---|---|---|---|-------|---|---|----|----|----|----|--------|----|----|----|--------|---|
| Hour | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | |
| | | | | | | | | | | | | | | | | | | 19 | |
| Line 1 | [| | | | | | Job 1 | | | |] | [| | Job 10 | |] | [| Job 11 |] |

²해법 2 도 완벽한 해결책은 아닙니다. 예를 들어, prg2com([7,8,9,10,11], 2) 의 이상적인 결과는 [[11,10], [9,8,7]] 이지만, 위에서 작성한 프로그램을 수행하면 [[11, 8, 7], [10, 9]] 라는 결과를 얻습니다. kds079 님께서 알려주셨습니다.

```
Line 3  [           Job 3           ] [           Job 8           ]  
Line 4  [           Job 4           ] [           Job 7           ]  
Line 5  [           Job 5           ] [           Job 6           ]
```

코드

- [ch10/prg2com.py](#)

A.3 진법 변환과 비트 연산

파이썬으로 진법 변환과 비트 연산 (bitwise operation) 을 하는 방법을 알아보려고 합니다. 조금 어려울 수도 있고 꼭 필요하지 않을 수도 있으니, 관심 있는 분만 보시면 됩니다.

진법 변환

이진수

파이썬에서 이진수 (binary) 는 `0b`를 앞에 붙여 나타냅니다.

```
>>> 0b0 # 이진수 0은 십진수 0  
0  
>>> 0b1 # 이진수 1은 십진수 1  
1  
>>> 0b10 # 이진수 10은 십진수 2  
2  
>>> 0b11 # 이진수 11은 십진수 3  
3
```

십진수를 이진수로 나타내고 싶다면 `bin()` 함수를 사용하면 됩니다.

```
>>> bin(0) # 십진수 0은 이진수 0  
'0b0'  
>>> bin(1) # 십진수 0은 이진수 1  
'0b1'  
>>> bin(2) # 십진수 2는 이진수 10  
'0b10'  
>>> bin(3) # 십진수 3은 이진수 11  
'0b11'
```

8 진수와 16 진수

비슷한 방법으로 십진수를 8 진수와 16 진수로도 변환할 수 있습니다. 하나씩 설명해드리지 않아도 아실 것 같으니, 반복문으로 한꺼번에 보여드릴게요.

```
>>> for i in range(20):
...     print(i, bin(i), oct(i), hex(i)) # 십진수, 이진수, 8진수, 16진수를 출력
...
0 0b0 0o0 0x0
1 0b1 0o1 0x1
2 0b10 0o2 0x2
3 0b11 0o3 0x3
4 0b100 0o4 0x4
5 0b101 0o5 0x5
6 0b110 0o6 0x6
7 0b111 0o7 0x7
8 0b1000 0o10 0x8
9 0b1001 0o11 0x9
10 0b1010 0o12 0xa
11 0b1011 0o13 0xb
12 0b1100 0o14 0xc
13 0b1101 0o15 0xd
14 0b1110 0o16 0xe
15 0b1111 0o17 0xf
16 0b10000 0o20 0x10
17 0b10001 0o21 0x11
18 0b10010 0o22 0x12
19 0b10011 0o23 0x13
```

불값의 논리 연산

비트 연산을 알아보기 전에 몸풀기로 불 (bool) 값의 논리 연산부터 알아볼게요. 불은 참 (True) 과 거짓 (False) 의 두 값만 있는 자료형입니다.

먼저 **and** 연산입니다. 왼쪽과 오른쪽 모두 True여야 결과도 True가 됩니다.

```
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
False
```

다음과 같이 진리표로도 나타내면 이해하기 쉽습니다.

| | | |
|-------|-------|-------|
| and | True | False |
| True | True | False |
| False | False | False |

다음으로 **or** 연산자입니다. 둘 중 하나라도 **True**이면 결과는 **True**가 됩니다. 둘 다 **False**일 때만 결과가 **False**가 됩니다.

```
>>> True or True
True
>>> True or False
True
>>> False or True
True
>>> False or False
False
```

진리표로 나타내면 다음과 같습니다.

| | | |
|-------|------|-------|
| or | True | False |
| True | True | True |
| False | True | False |

and와 **or** 연산자는 두 개의 항 (피연산자) 을 갖는다는 뜻으로 '이항 (二項, binary)'로 분류합니다.

이번에는 **not** 연산자를 알아볼게요. **not True**는 **False**, **not False**는 **True**입니다.

```
>>> not True
False
>>> not False
True
```

not 연산자는 '단항 (單項, unary) 연산자'입니다.

불과 정수의 관계

다음으로 파이썬에서 불값과 정수값의 관계를 알아보겠습니다.

파이썬에서 `True`는 숫자 1과 같습니다.

```
>>> True == 1 # 등호가 두 개임에 유의(할당 연산이 아니라 비교)
True
```

그리고 `False`는 숫자 0과 같습니다.

```
>>> False == 0
True
```

비트 연산

&는 비트 AND 연산자로, 앞에서 살펴본 `and`와 비슷합니다. 그리고 |는 비트 OR 연산자로, `or`와 비슷합니다.

한 자리 이진수의 비트 연산

우선 아주 작은 수부터 계산해볼게요. 이진수 1(`0b1`) 과 이진수 0(`0b0`)입니다.

비트 AND 연산 먼저 비트 AND(&) 연산입니다.

```
>>> bin(0b1 & 0b1)
'0b1'
>>> bin(0b1 & 0b0)
'0b0'
>>> bin(0b0 & 0b1)
'0b0'
>>> bin(0b0 & 0b0)
'0b0'
```

비트 OR 연산 다음은 비트 OR(|) 연산입니다.

```
>>> bin(0b1 | 0b1)
'0b1'
>>> bin(0b1 | 0b0)
'0b1'
>>> bin(0b0 | 0b1)
'0b1'
>>> bin(0b0 | 0b0)
'0b0'
```

비트 NOT 연산 비트 NOT(~) 연산도 해볼까요?

```
>>> bin(~0b1)  
'-0b10'
```

결과가 음수로 나왔네요.

~ 연산 결과는 원래 수에 마이너스 기호를 붙인 뒤 1을 뺀 값과 같습니다.

```
>>> bin(~0b0)  
'-0b1'  
>>> bin(~0b0 - 0b1)  
'-0b1'
```

위에서 두 연산의 결과가 같은 것을 확인할 수 있습니다.

XOR 연산 XOR(배타적 OR, exclusive OR) 이라는 연산 (^) 도 있습니다. 두 값이 모두 1 이거나 모두 0 이면 결과가 0이고, 두 값 중 하나는 1이고 다른 하나가 0이면 결과가 1이 됩니다.

```
>>> bin(0b1 ^ 0b1)  
'0b0'  
>>> bin(0b1 ^ 0b0)  
'0b1'  
>>> bin(0b0 ^ 0b1)  
'0b1'  

```

그 외의 비트 연산자는 뒤에서 알아볼게요.

네 자리 이진수의 비트 연산

한 자리의 비트 연산자를 이해했으면 네 자리 이진수로 비트 연산을 해 보겠습니다. 우선 & 연산입니다.

```
>>> bin(0b1111 & 0b1100)  
'0b1100'  
>>> bin(0b1010 & 0b1100)  
'0b1000'
```

비트 연산은 한 자리씩 따로따로 계산을 합니다. 위의 결과를 잘 살펴보세요.

다음은 | 연산입니다.

```
>>> bin(0b1111 | 0b1100)
'0b1111'
>>> bin(0b1010 | 0b1100)
'0b1110'
```

~ 연산도 해 보겠습니다.

```
>>> bin(~0b1100)
'~0b1101'
>>> bin(~0b1001)
'~0b1010'
```

^ 연산은 어떻게 될지 궁금하군요.

```
>>> bin(0b1111 ^ 0b1100)
'0b11'
>>> bin(0b1010 ^ 0b1100)
'0b110'
```

시프트 연산

이번에는 시프트 (shift) 연산을 알아보겠습니다.

먼저 왼쪽 시프트 (<<) 입니다.

```
>>> bin(0b1 << 1)
'0b10'
>>> bin(0b1 << 2)
'0b100'
>>> bin(0b1 << 3)
'0b1000'
```

위에서 1이 점점 왼쪽으로 이동하는 것을 볼 수 있습니다.

다음은 오른쪽 시프트 (>>) 입니다.

```
>>> bin(0b1010 >> 1)
'0b101'
```

1이 오른쪽으로 한 자리씩 밀린 것을 볼 수 있습니다. 더 많이 시프트하면 어떻게 될까요?

```
>>> bin(0b1010 >> 2)
'0b10'
```

```
>>> bin(0b1010 >> 3)
'0b1'
>>> bin(0b1010 >> 4)
'0b0'
```

점점 밀려서 결국 0이 되었네요.

십진수의 비트 연산

그런데 비트 연산을 이진수에만 할 수 있는 것은 아니랍니다. 앞에서 이진수로 했던 계산을 십진수로 다시 해봐도 결과가 같은 것을 볼 수 있습니다.

```
>>> 0b1111
15
>>> 0b1100
12
>>>
>>> 15 & 12
12
>>> bin(12)
'0b1100'
```

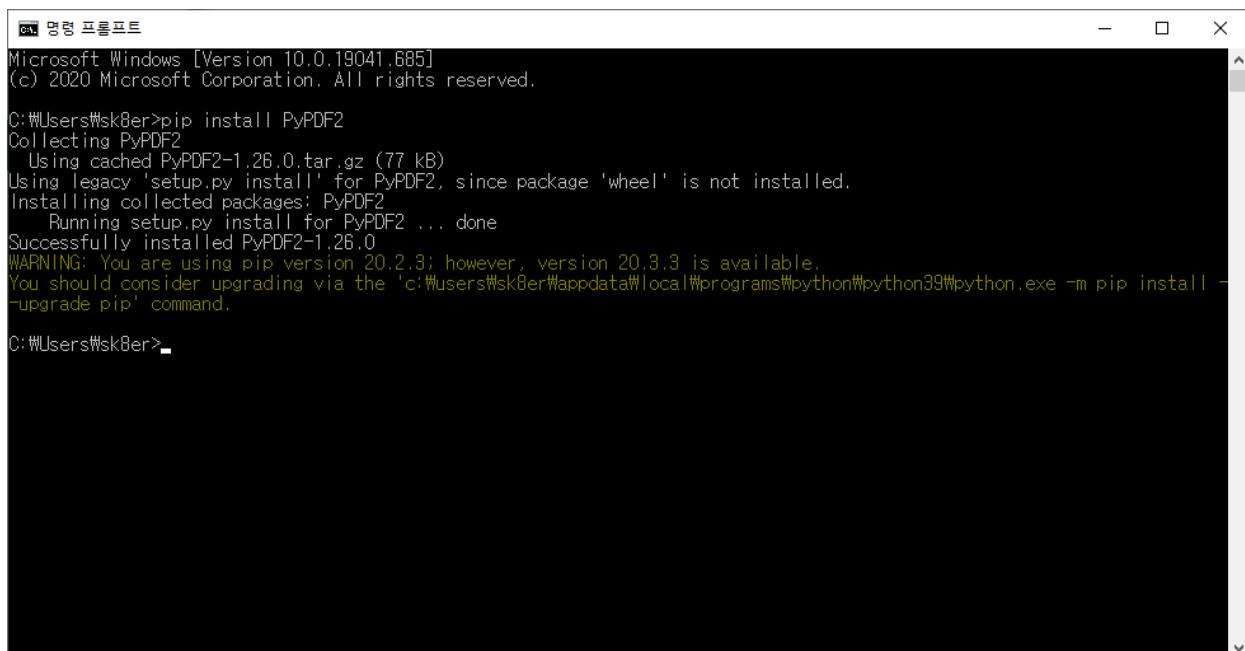
그도 그럴 것이, 겉으로 보기에 이진수로 나타내든 십진수로 나타내든, 컴퓨터는 이진수를 가지고 계산하고 있을 테니까요.

A.4 파일로 PDF 파일 합치기

PDF 문서를 태블릿으로 읽을 때가 많은데, PDF 파일 여러 개를 하나로 합치고 싶을 때가 종종 있더라구요. 파일을 이용해 간단히 처리할 수 있습니다.

PyPDF2 설치

이 장에서 소개하는 예제를 실행하려면 [PyPDF2 패키지](#)라는 패키지가 필요합니다. 명령 프롬프트나 리눅스 셸에서 `pip install PyPDF2` 명령으로 설치해주세요.



The screenshot shows a Windows Command Prompt window titled "명령 프롬프트". The window displays the output of a pip command to install PyPDF2. The text in the window is as follows:

```
Microsoft Windows [Version 10.0.19041.685]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\sk8er>pip install PyPDF2
Collecting PyPDF2
  Using cached PyPDF2-1.26.0.tar.gz (77 kB)
  Using legacy 'setup.py install' for PyPDF2, since package 'wheel' is not installed.
  Installing collected packages: PyPDF2
    Running setup.py install for PyPDF2 ... done
Successfully installed PyPDF2-1.26.0
WARNING: You are using pip version 20.2.3; however, version 20.3.3 is available.
You should consider upgrading via the 'c:\Users\sk8er\AppData\Local\Programs\Python\Python39\python.exe -m pip install --upgrade pip' command.

C:\Users\sk8er>
```

PDF 파일 준비

예제를 실행하려면 병합할 대상이 되는 PDF 파일이 필요합니다. 갖고 계신 PDF 파일을 복사해서 `mybook_01.pdf`, `mybook_02.pdf`, `mybook_03.pdf`, `mybook_04.pdf`라는 이름으로 준비해 주세요.

첫 번째 버전

그럼 가장 단순한 버전의 코드를 보여드릴게요.

```
from PyPDF2 import PdfFileMerger

pdfs = ['mybook_01.pdf', 'mybook_02.pdf', 'mybook_03.pdf', 'mybook_04.pdf']

merger = PdfFileMerger()

for pdf in pdfs:
    merger.append(pdf)

merger.write("mybook.pdf")
merger.close()
```

위 코드를 실행하면 파일이 실행되는 디렉터리에 있는 네 개의 PDF 파일 `mybook_01.pdf`, `mybook_02.pdf`, `mybook_03.pdf`, `mybook_04.pdf`가 순서대로 합쳐진 `mybook.pdf`라는 파일이 만들어집니다.

두 번째 버전

그런데 합치고 싶은 파일이 꼭 네 개만 있으라는 법은 없죠. 그래서 앞에서 배운 [glob 모듈](#)을 사용해서 파일명이 `mybook`으로 시작하는 모든 PDF 파일을 합치도록 개선해보았습니다.

```
from glob import glob
from PyPDF2 import PdfFileMerger

BOOK = 'mybook'

merger = PdfFileMerger()

for f in glob('*' + BOOK + '*.pdf'):
    merger.append(f)
```

```
merger.write(BOOK + ".pdf")
merger.close()
```

이제 처음보다는 좀 더 쓸만한 스크립트가 되었습니다. 하지만 계속 쓰다 보니 몇 가지 불편한 점이 더 있더라구요.

스크립트가 있는 폴더와 같은 곳에 있는 파일만 합칠 수 있기 때문에 다른 파일들을 합치려면 스크립트를 복사해야 하고, 파일명도 스크립트에 들어있다보니 그때그때 수정해야 합니다.

병합된 파일의 이름을 원본과 같은 이름으로 지을 경우, 스크립트를 두 번 이상 실행하면 처음 실행할 때 병합된 파일이 두 번째 실행 때 입력으로 들어가는 문제점도 있었구요.

세 번째 버전

그래서 이번에는 병합할 대상 파일이 있는 디렉터리와 책 제목을 스크립트를 실행할 때 인자로 받을 수 있게 바꿔봤습니다. 제가 현재 실제로 사용하는 스크립트입니다.

```
import argparse
from glob import glob
import os

from PyPDF2 import PdfFileMerger

def main(book_title, directory="."):
    merger = PdfFileMerger()

    for f in glob(f"{directory}/{book_title}*.pdf"):
        merger.append(f)

    os.chdir(directory)
    if not os.path.isdir(sub_dir):
        os.mkdir(sub_dir)

    merger.write(f"{directory}/{sub_dir}/{bookname}.pdf")
    merger.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("-d", "--directory", help="directory where files to be merged live")
    parser.add_argument("bookname")
    args = parser.parse_args()
    directory = args.directory
    bookname = args.bookname

    main(args.bookname, args.directory)
```

왕초보를 위한 Python: 쉽게 풀어 쓴 기초 문법과 실습

인자를 입력받는 기능을 위해 파이썬에서 기본으로 제공하는 `argparse` 모듈을 이용했습니다.

이 스크립트 (`pdf_merge.py`)를 아무 인자 없이 실행하면 다음과 같이 사용법과 오류 메시지가 출력됩니다.

```
>python pdf_merge.py
usage: pdf_merge.py [-h] [-d DIRECTORY] bookname
pdf_merge.py: error: the following arguments are required: bookname
```

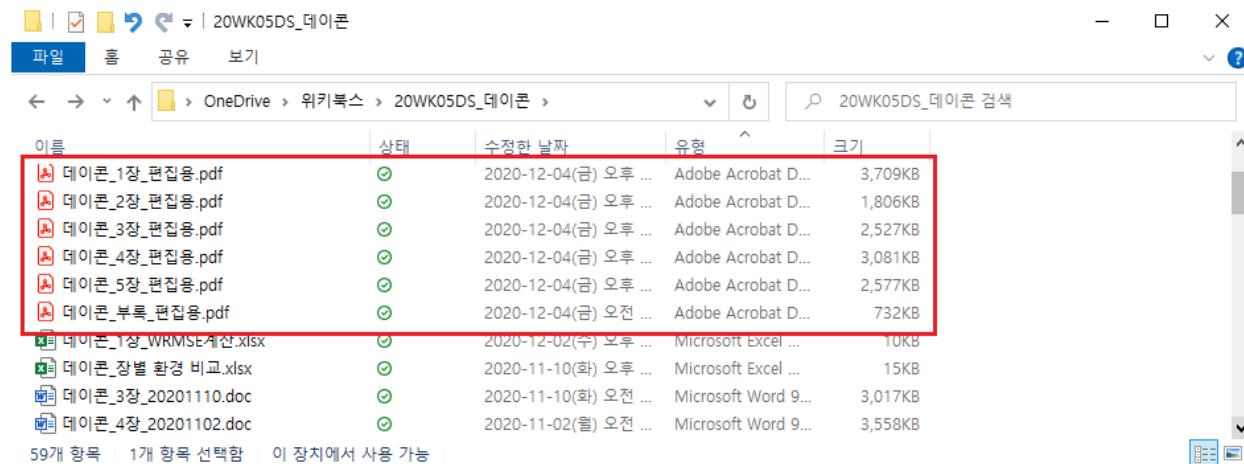
이 스크립트의 사용법은 다음과 같습니다.

```
python pdf_merge.py -d <디렉터리> <책 이름>
```

사용 예

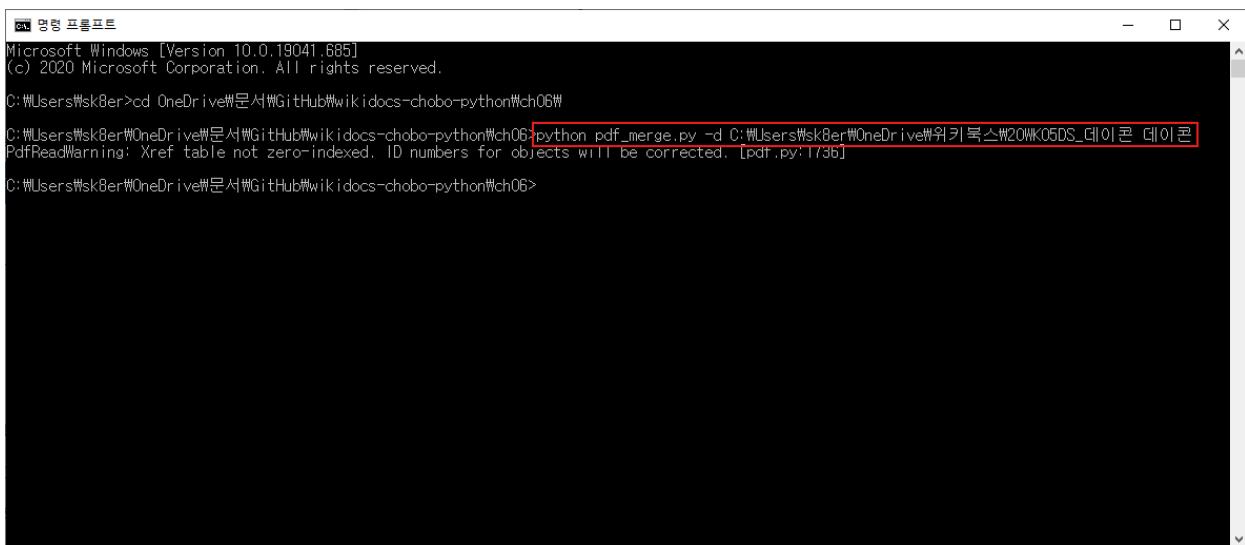
이 스크립트를 실제로 사용한 예를 보여드릴게요.

다음 그림의 윈도우 탐색기에는 합치고자 하는 원본 PDF 파일들이 보입니다.



다음 그림과 같이 명령 프롬프트에서 `pdf_merge.py`를 실행합니다.

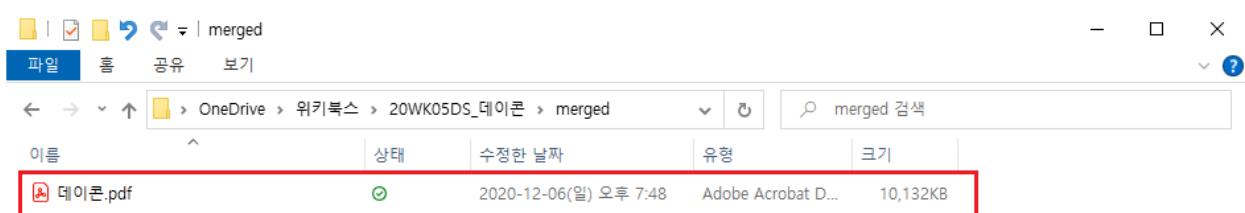
왕초보를 위한 Python: 쉽게 풀어 쓴 기초 문법과 실습



```
명령 프롬프트
Microsoft Windows [Version 10.0.19041.685]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\sk8er>cd OneDrive\문서\GitHub\wikidocs-chobo-python\ch06
C:\Users\sk8er\OneDrive\문서\GitHub\wikidocs-chobo-python\ch06>python pdf_merge.py -d C:\Users\sk8er\OneDrive\위키북스\20WK05DS_데이터\데이터
PdfReadWarning: Xref table not zero-indexed. ID numbers for objects will be corrected. [pdf.py:1736]
C:\Users\sk8er\OneDrive\문서\GitHub\wikidocs-chobo-python\ch06>
```

실행 결과, 원본 PDF 파일들을 병합한 파일이 merged 디렉터리에 만들어졌습니다.



개선할 점

이 스크립트는 제가 혼자 사용하는 것이라 도움말도 친절하지 않고 기능도 만들다 말았습니다. 출력할 디렉터리명은 main() 함수의 sub_dir 인자에 `merged`라고 지정했는데, 이것을 바꾸고 싶다면 argparse 를 이용해 인자를 추가해서 main 함수에 넘겨줄 수도 있겠죠. 처음에 그렇게 할 생각이었는데, 별로 필요하지 않아서 완성하지 않았습니다.

여러분이 사용하다가 불편한 점이 있으면 직접 수정해보세요.

A.5 matplotlib 으로 하트 그리기

matplotlib 패키지

matplotlib 은 플롯 (그래프) 을 그릴 때 많이 쓰이는 파이썬 라이브러리입니다. matplotlib 프로젝트의 홈페이지 주소는 다음과 같습니다.

- matplotlib 페이지 <http://sourceforge.net/projects/matplotlib/>

파이썬 셸에서 다음 명령을 실행해보세요.

```
from pylab import *
```

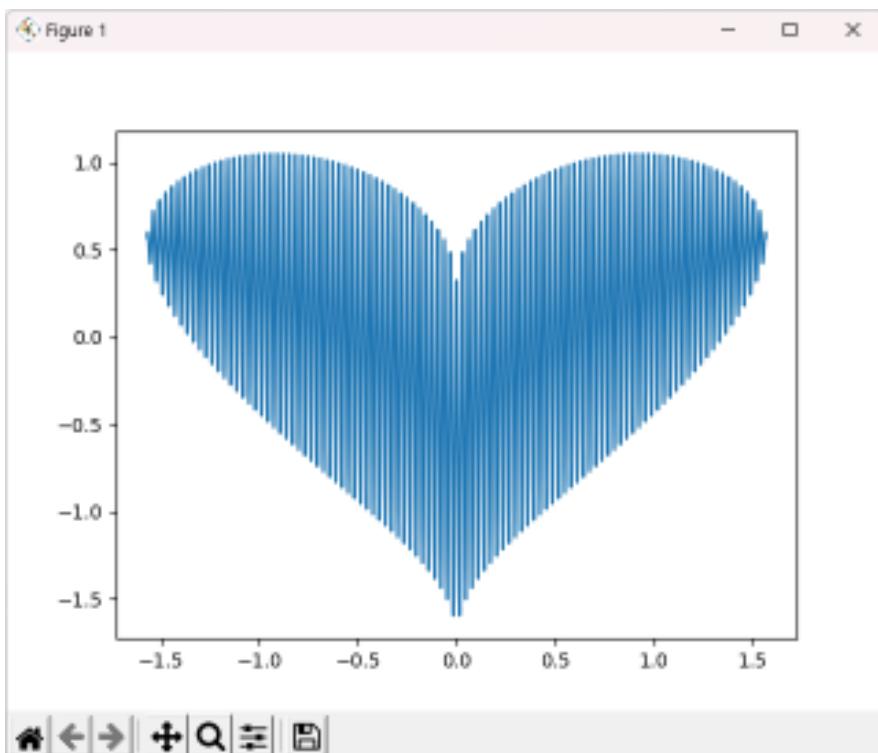
ModuleNotFoundError: No module named 'pylab'이라는 오류 메시지가 나온다면 파이썬 셸에서 빠져나와서 pip install matplotlib 명령 (아나콘다 환경에서는 conda install matplotlib) 으로 matplotlib을 설치합니다.

하트 그리기

하트를 그려볼게요 ~

```
1 from pylab import *
2 x = linspace(-1.6, 1.6, 10000)
3 f = lambda x: (sqrt(cos(x)) * cos(200 * x) + sqrt(abs(x)) - 0.7) * \
4     pow((4 - x * x), 0.01)
5 plot(x, list(map(f, x)))
6 show()
```

실행하면 예쁜 하트가 그려집니다.



코드 설명

2행: `linspace()`는 주어진 숫자 범위 내에서 동일한 간격으로 떨어진 숫자들을 쉽게 만들어줍니다.

```
>>> import pylab
>>> pylab.linspace(5, 7, 20)
array([ 5.          ,  5.10526316,  5.21052632,  5.31578947,  5.42105263,
       5.52631579,  5.63157895,  5.73684211,  5.84210526,  5.94736842,
       6.05263158,  6.15789474,  6.26315789,  6.36842105,  6.47368421,
       6.57894737,  6.68421053,  6.78947368,  6.89473684,  7.        ])
```

3~4 행: 람다 함수를 만들어서 `f`라는 이름을 붙여주었습니다. 함수 내용이 너무 길어서 역슬래시를 사용해서 두 줄에 나누어 썼습니다. `lambda` 와 `map`에 대해서는 [3.5. 람다의 설명](#)을 참고하세요.

`pow()`는 제곱을 구하는 내장 (built-in) 함수입니다.

```
>>> pow(2, 3)
8
```

`sqrt()`는 제곱근을, `cos()`는 코사인을 구하는 함수로, `pylab`에서 임포트했습니다.

5행: `plot()`은 플롯을 만들어주고,

6 행: `show()`를 하면 화면에 보여집니다.

`matplotlib`의 사용법을 익히면 다양한 형태의 플롯을 그릴 수 있고, 제목이나 x 축, y 축의 라벨을 붙이는 것도 손쉽게 할 수 있답니다.

- Python 활용: 이미지 처리와 데이터 분석 <https://www.slideshare.net/sk8erchoi/anaconda-50854172>

A.6 윈도우 CMD에서 파이썬 활용 팁

윈도우 운영체제의 명령 프롬프트 (CMD)에서 파이썬을 편리하게 활용하는 팁을 소개합니다.

파이썬 런처 (Python Launcher)

명령행에서 파이썬 스크립트를 호출하는 한 가지 방법은 파이썬 인터프리터 경로와 파이썬 스크립트 파일명을 입력하는 것입니다. 예를 들어 파이썬이 <C:\Users\ychoi\AppData\Local\Programs\Python\Python39>에 설치됐다면, 다음과 같이 복잡한 명령을 실행합니다 (맨 앞의 >는 윈도우 프롬프트 문자열을 뜻하는 것으로 입력하지 않습니다).

```
> C:\Users\ychoi\AppData\Local\Programs\Python\Python39\python.exe hello.py
```

위의 방법은 아무래도 번거로운데요, [python.org](https://www.python.org)에서 윈도우용 파이썬을 다운로드해 설치했다면 함께 설치된 파이썬 런처 (Python Launcher)를 이용해 다음과 같이 간편하게 실행할 수 있습니다.

```
> py hello.py
```

또는 간단히 스크립트 경로만 입력해 실행할 수도 있고요.

```
> hello.py
```

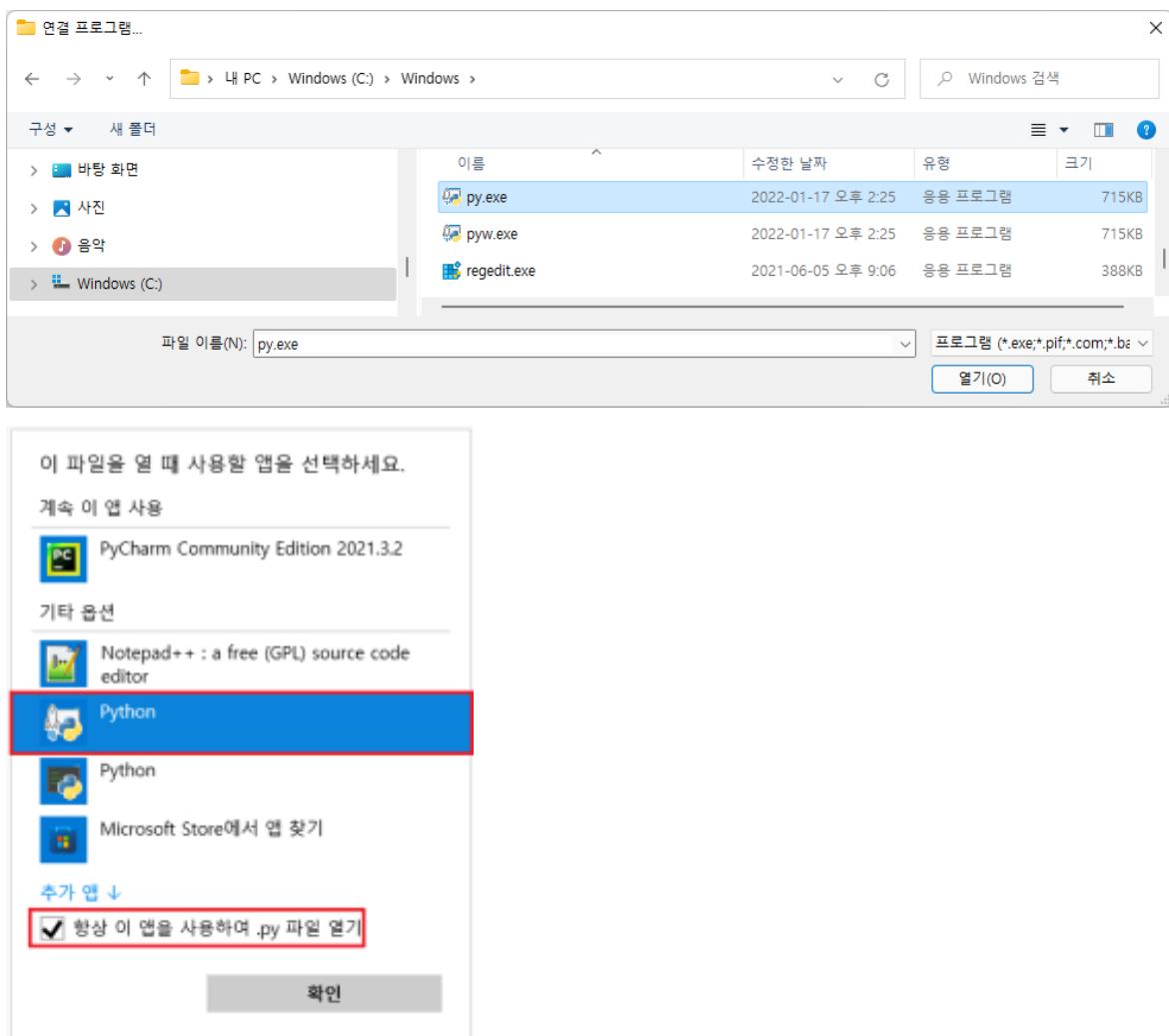
또한 파이썬 런처에는 파이썬 스크립트의 첫 번째 행에 지정된 파이썬 인터프리터를 찾아 실행해주는 기능도 있습니다.

```
#!/usr/bin/python3
```

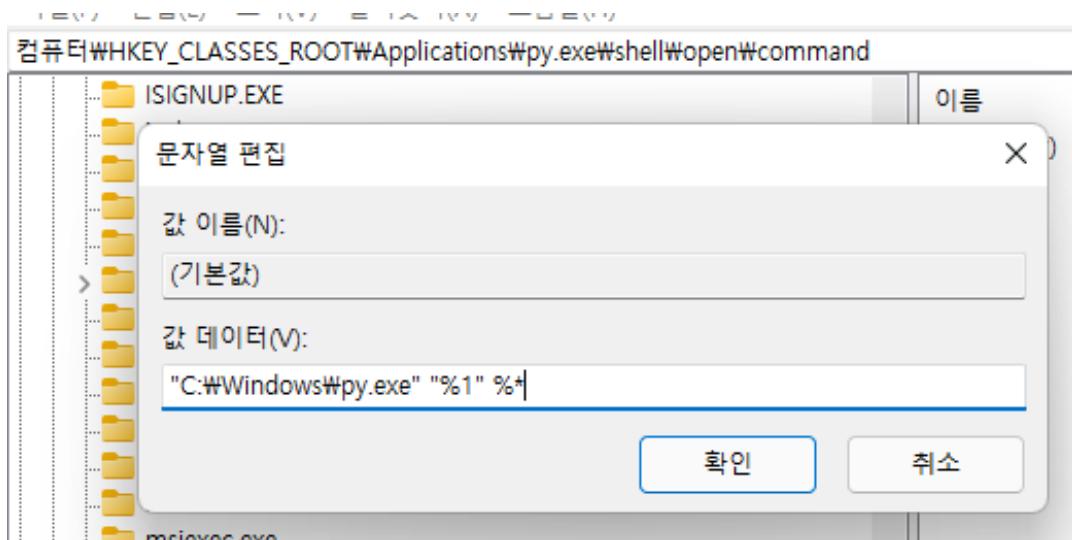
스크립트의 첫 행이 위와 같이 되어 있다면, 시스템에 설치된 파이썬 3 인터프리터를 사용해 스크립트를 실행해줍니다.

::: box 윈도우 명령 프롬프트에서 파이썬 스크립트 실행 시 인자가 제대로 전달되지 않을 때 해결 방법

1. 설정 > 앱 > 기본 앱 > 파일 유형별 기본값 선택에서 .py를 선택합니다.
2. 앞으로.py 파일을 열 때 사용할 방법을 선택하세요 창에서 이 PC에서 다른 앱 찾기를 선택하고 C:\Windows\py.exe를 지정합니다.



3. 그래도 해결되지 않는다면 [윈도우 레지스트리를 수정하는 방법](#)을 시도할 수 있습니다.



...

Path 환경변수

명령행에서 파이썬 스크립트를 실행할 때는 스크립트가 있는 디렉터리 (폴더)로 이동해서 실행하거나, 경로를 지정해야 합니다.

스크립트가 있는 디렉터리로 이동해 실행하는 예 (C:\GitHub\ychoi\utils에 txt_merge.py 파일이 있다고 가정):

```
C:\Users\ychoi>cd ..\..
C:\>cd GitHub\ychoi\utils
C:\GitHub\ychoi\utils>txt_merge.py
```

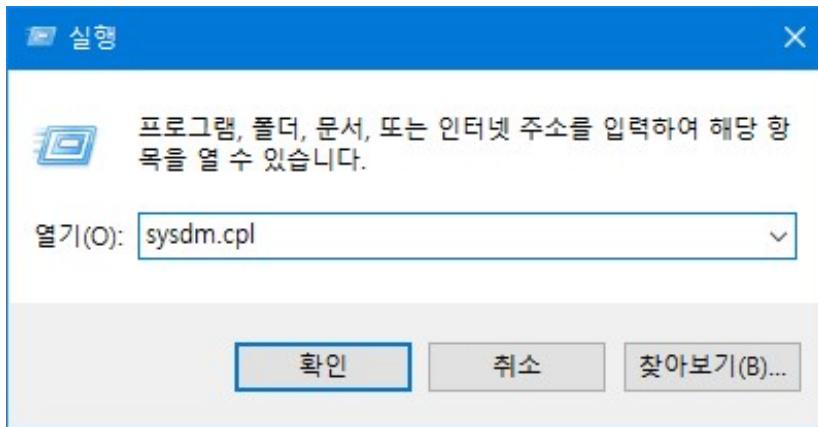
경로를 지정해 실행하는 예:

```
C:\Users\ychoi>\GitHub\ychoi\utils\txt_merge.py
```

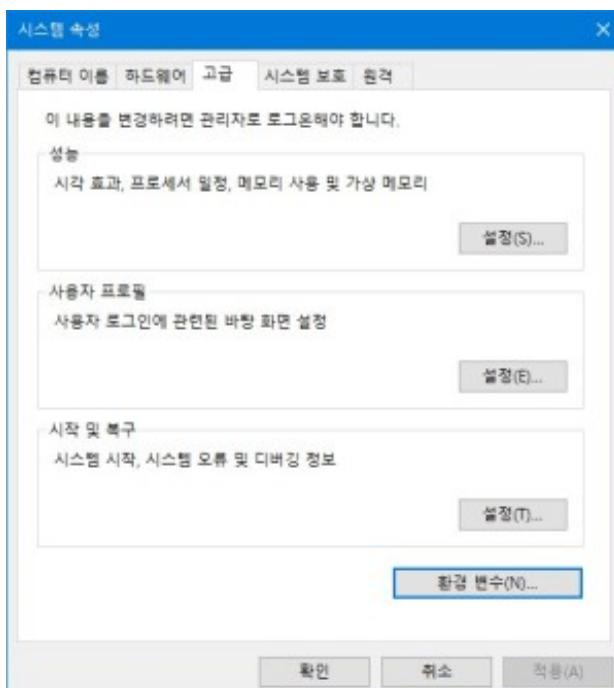
위와 같이 디렉터리를 찾아가거나 경로를 입력하기 번거롭다면, 스크립트가 있는 곳을 PATH 환경변수에 등록해두면 스크립트 파일명만으로 실행할 수 있어 편리합니다.

방법은 다음과 같습니다.

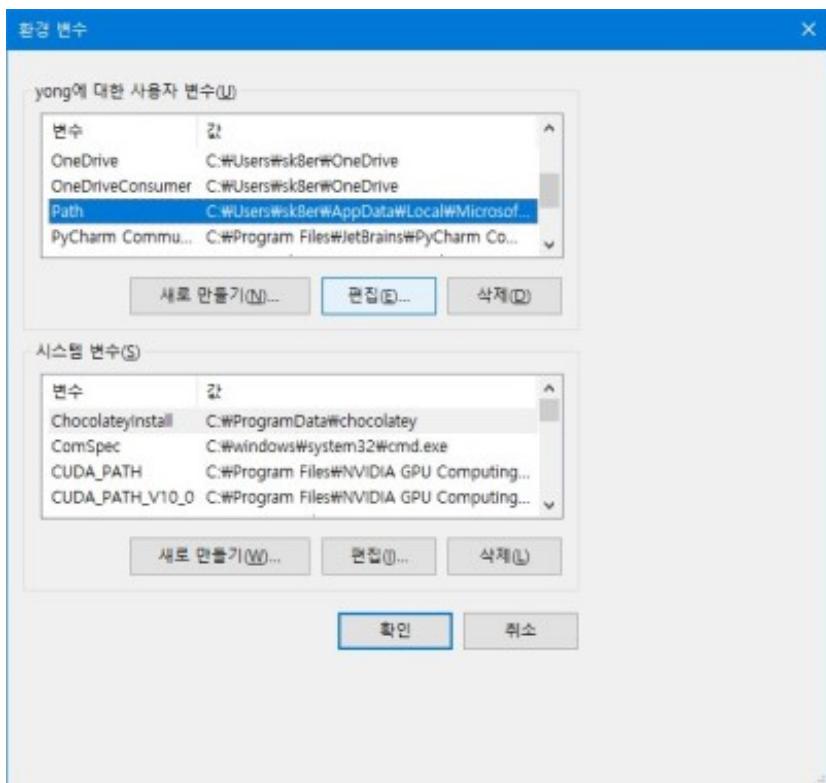
1. + r 키를 누르고 sysdm.cpl 입력 후 확인 버튼을 클릭



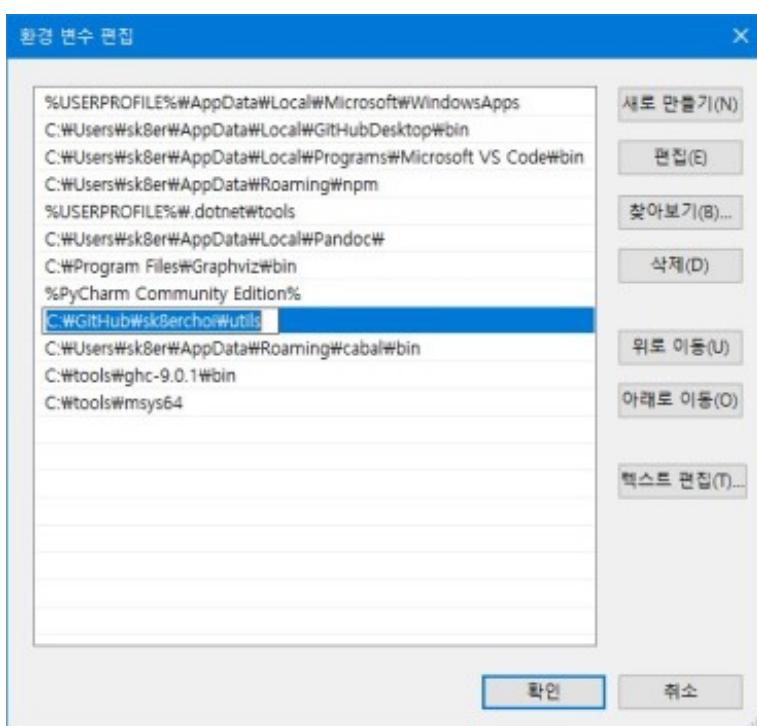
2. 시스템 속성 창의 고급 탭에서 환경 변수 버튼을 클릭



3. 사용자 변수에서 Path를 선택하고 편집 버튼을 클릭



4. 환경 변수 편집 창의 새로 만들기 버튼을 클릭하고 스크립트가 있는 디렉터리를 입력하고 확인 버튼을 클릭

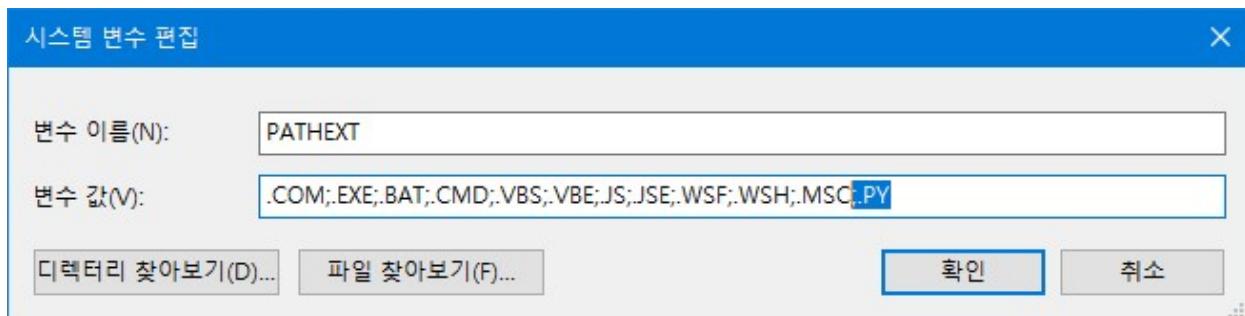


이제 명령 프롬프트 창을 닫고 새로운 명령 프롬프트 창을 열면 어느 위치에서나 (Path에 등록된 경로의) 스크립트를 실행할 수 있습니다.

```
C:\Users\ychoi>txt_merge.py
```

PATHEXT 환경변수

파이썬 스크립트를 확장자 (.py) 없이 호출하고 싶다면 시스템 PATHEXT 환경변수에 .PY를 등록합니다. (기존 환경변수의 맨 뒤에 ;를 붙이고 .PY를 입력합니다.)



이제 명령 프롬프트 창을 닫고 새로운 창을 열면 다음과 같이 확장자 없이 스크립트를 실행할 수 있습니다.

```
>txt_merge
```

A.7 일회용 스크립트를 재사용 가능하게 바꾸기

우리는 귀찮은 작업을 편하게 처리하려고, 또는 수작업을 하다가 실수를 하지 않으려고 스크립트를 작성하죠.

그런데 가끔은 딱 한 번만 쓰고 말 생각으로, 또는 단 한 가지 용도로 스크립트를 작성하기도 합니다. 아래 스크립트가 그 예입니다.

파일명: `wikidocs_toc_chobopython.py`

```
1 import urllib.request
2 from bs4 import BeautifulSoup
3
4
5 url = 'https://wikidocs.net/book/2'
6 with urllib.request.urlopen(url) as f:
7     html = f.read().decode('utf-8')
8
9 soup = BeautifulSoup(html, 'html.parser')
10 titles = soup.select('.list-group-item > span')
11 for title in titles[1:]:
12     s = title.select('span')[0].text.strip()
13     print(s)
```

이 스크립트를 실행하면 위키독스의 《왕초보를 위한 파일》 (<https://wikidocs.net/book/2>) 목차가 다음과 같이 출력됩니다.

```
$ python3 wikidocs_toc_chobopython.py
0 머리말
0.1 주요 변경 이력
1 파일 시작하기
1.1 파일 맛보기
1.2 변수
1.2.1 연습 문제: 파일 크기 계산
...
A.4 파일으로 PDF 파일 합치기
A.5 matplotlib으로 하트 그리기
A.6 윈도우 CMD에서 파일 활용 팁
Bye~
```

이 스크립트를 작성할 때만 해도 이 책 외에 다른 책의 목차를 출력할 일이 없을 것으로 생각했는데, 나중에 그럴 필요가 생겼다면 어쩌면 좋을까요? 예를 들어, 《SQLite3로 가볍게 배우는 데이터베이스》(<https://wikidocs.net/book/1530>)의 목차를 출력하고 싶다면요?

가장 쉬운 방법은 스크립트를 복사해서 새로운 파일 (예: `wikidocs_toc_sqlite3.py`) 을 만든 뒤, 새로 만든 스크립트의 5 번째 줄을 다음과 같이 수정하는 것이겠죠.

```
url = 'https://wikidocs.net/book/1530'
```

이 스크립트를 실행하면 다음 결과가 출력됩니다.

```
$ python3 wikidocs_toc_sqlite3.py
A01 도 입
A02 실습 환경 갖추기
A03 데 이터 베이스와 테이블 만들기
A04 추가, 삭제, 갱신, 조회 (1)
...
B03 실전 예제 - 전기차 충전소 데이터 정제
B04 실전 예제 - 새의 GPS 데이터 가공(1)
B05 실전 예제 - 새의 GPS 데이터 가공(2)
```

현재 코드

현재 코드는 다음 링크에서 확인할 수 있습니다.

- [wikidocs_toc_chobopython.py](#)
- [wikidocs_toc_sqlite3.py](#)

스크립트의 사용성을 개선하는 과정

그런데 또 다른 책의 목차를 알고 싶어지면, 그때는 어떻게 해야 할까요? 기능이 거의 비슷하지만 똑같지는 않은 스크립트가 필요해지면, 새로운 스크립트를 작성해야만 할까요?

이런 일을 다음에 또 해야 한다면, 하루에도 여러 번 해야 한다면, 그때마다 스크립트를 수정해서 저장한 다음 실행하는 것조차 귀찮은 일이 되어버립니다.

그렇다면, 단 한 가지 일만 처리하는 스크립트 여러 개를 작성하는 것보다, 스크립트 하나로 여러 가지 일을 처리할 수 있게, 즉 ‘재사용’이 가능하게, 이전에 만들어둔 스크립트를 손질하는 것이 좋을 수도 있습니다.

이럴 때 스크립트을 어떻게 바꿔 나가는 것이 좋을지를 단계별로 소개하려고 합니다. 모든 단계를 끝내야 하는 것이 아니라, 한 단계를 마칠 때마다 바로 사용할 수 있으면서도 이전보다는 조금씩 더 편리해질 것입니다. 여기서 소개하는 순서나 방법이 반드시 옳다기보다는, 이런 식으로도 할 수 있다는 예시로 이해해주시면 되겠습니다.

1. 실행 가능하게 만들기
2. 대상을 인자로 받기
3. 명령을 인자로 받기
4. 기능 추가
5. 도움말 추가

A.7.1 실행 가능하게 만들기

목표

터미널 (또는 명령 프롬프트)에 `python3 wikidocs_toc_chobopython.py`를 타자하는 대신 스크립트 파일명인 `wikidocs_toc_chobopython.py`만 타자해서 실행하게 만드는 것이 이번 단계의 목표입니다. 스크립트를 한번 실행할 때마다 타자할 글자 수를 7~8 글자 줄이는 효과가 있죠.

절차

스크립트에 실행 권한 (execute permission)을 부여합니다.

예:

```
$ chmod +x wikidocs_toc_chobopython.py
```

첫 번째 줄에 다음 행을 추가합니다. 이런 것을 ‘시뱅 (shebang)’이라고 부릅니다.

```
#!/usr/bin/python3
```

커밋 로그

- [d454ebe](#)

현재 코드

- [wikidocs_toc_chobopython.py](#)
- [wikidocs_toc_sqlite3.py](#)

스크립트 사용 예

이제 터미널에서 `python` 또는 `python3`를 없이, 스크립트 파일명만 타자해서 스크립트를 실행할 수 있습니다.

```
$ wikidocs_toc_chobopython.py
0. 머리말
0.1 주요 변경 이력
1. 파일 썬 시작하기
1.1 파일 썬 맛 보기
...
```

```
$ wikidocs_toc_sqlite3.py
A01 도입
A02 실습 환경 갖추기
A03 데이터베이스와 테이블 만들기
A04 추가, 삭제, 갱신, 조회 (1)
...
```

A.7.2 대상을 인자로 받기

목표

지금은 책의 URL이 하드 코딩되어 있어서, 다른 책의 목차를 출력하고 싶으면 그때마다 스크립트를 열어서 수정해야 하는 불편이 있습니다. 스크립트를 실행할 때 책 번호 (예: 왕초보를 위한 파이썬의 URL <https://wikidocs.net/book/2>에서 책 번호는 2)를 인자로 받을 수 있게 함으로써, 스크립트를 매번 수정하지 않고도 다른 책의 목차를 얻을 수 있게 바꾸는 것이 이번 단계의 목표입니다.

절차

파이썬에서 명령행의 인자를 알아내는 가장 원초적 (?) 인 방법은 sys 모듈의 argv를 사용하는 것입니다.

스크립트의 앞부분에 다음 문장을 추가합니다.

```
import sys
```

하드코딩했던 것을 인자값을 전달받도록 수정합니다.

```
url = 'https://wikidocs.net/book/' + sys.argv[1]
```

이제 새로운 책의 목차를 얻을 때마다 스크립트를 수정하거나 책마다 스크립트를 따로 만들 필요가 없어졌습니다.

스크립트 이름은 `wikidocs_toc_chobopython.py`에서 `wikidocs_toc.py`로 바꾸는 것이 좋겠습니다. 그리고 `wikidocs_toc_sqlite3.py`는 이제 필요 없으므로 삭제합니다.

커밋 로그

- [376573d](#)

현재 코드

두 개였던 스크립트가 하나로 줄었습니다.

- [wikidocs_toc.py](#)

스크립트 사용 예

```
$ wikidocs_toc.py 6038
강좌 소개
0. 탐색
0.1 탐색 문제 정의
0.2 탐색 문제 풀기
0.3 무정보(uninformed) 탐색 - 깊이 우선(depth-first), 너비 우선(breadth-first)
  0.3.1 DFS와 BFS의 장단점
  ...
  
```

A.7.3 명령을 인자로 받기

책의 목차를 얻어오는 스크립트를 만들어서 잘 쓰고 있었는데, 이번에는 본문 내용을 가져올 일이 생겼습니다. 본문만을 처리하는 스크립트를 따로 만들지 않고, 기존 스크립트에 기능을 추가하고 싶습니다.

이 기능은 (1) 명령을 인자로 받기와 (2) 새로운 명령을 추가하기의 두 단계로 구현합니다.

목표

명령을 받는 인자를 추가합니다.

절차

인자:

- 첫 번째 인자: 명령 ([toc](#))
- 두 번째 인자: 책 번호

파일명을 [wikidocs_toc.py](#)에서 [wikidocs.py](#)로 바꿉니다.

커밋 로그

- [8c72b45](#)

현재 코드

- [wikidocs.py](#)

스크립트 사용 예

```
$ wikidocs.py toc 2
0. 머리말
0.1 주요 변경 이력
1. 파일 썬 시작하기
...
```

A.7.4 기능 추가

목표

본문 내용을 출력하는 기능을 추가합니다.

절차

책 내용을 받아오는 명령을 추가합니다.

인자:

- 첫 번째 인자: 명령 (`toc` 또는 `content`)
- 두 번째 인자: 책 번호 또는 페이지 번호

커밋 로그

- [2dff0a4](#)

현재 코드

- [wikidocs.py](#)

스크립트 사용 예

```
$ wikidocs.py content 43
```

```
프로그래밍은 재미있습니다. 진짜로 재미있습니다.  
그 재미를 알려면 프로그래밍을 직접 해봐야하는데, 사실 프로그래밍을 시작하는 것이  
쉽지가 않습니다. 그래서 여러분과 함께 배우기 쉬운  
파이썬이라는 언어를 함께 공부해보려고 합니다.
```

...

A.7.5 도움말 추가

스크립트에 도움말 (help) 기능이 있으면 다른 사람이 활용하는 데 도움이 됩니다. 스크립트를 만든 본인도 시간이 흐르면 사용법이 잘 생각나지 않아서 코드를 들여다보게 되니, 자신에게도 도움이 됩니다.

목표

스크립트 사용법을 알려주는 help 옵션을 추가합니다.

절차

도움말을 작성합니다.

도움말을 출력하는 명령을 추가합니다.

예에서는 sys.argv 를 사용했지만, argparse를 사용해도 좋습니다.

커밋 로그

- [4a53548](#)

현재 코드

- [wikidocs.py](#)

스크립트 사용 예

```
$ wikidocs.py  
Number of arguments is not matched  
Try "python wikidocs.py help".
```

```
$ wikidocs.py help  
Usage:  
python wikidocs.py <command> [content id]  
  
Description:  
Available commands are toc, content and help.  
Content id should be a number.  
  
Examples:  
python wikidocs.py toc 2  
python wikidocs.py content 43
```

Bye~

끝까지 완주하신 여러분 축하합니다!

그리고 읽어주셔서 고맙습니다. 느끼신 점을 댓글로 남겨주시면 좋겠습니다.

파이썬을 좀 더 알고 싶다면 제가 번역한 [《실용 파이썬 프로그래밍》](#) 도 한번 읽어보세요.

