



**초보자를 위한
Python 200제**

초판 1쇄 인쇄 | 2017년 2월 22일
초판 1쇄 발행 | 2017년 2월 27일

지 은 이 | 장삼용
발 행 인 | 이상만
발 행 처 | 정보문화사

책임 편집 | 최동진
편집 진행 | 노미라

주 소 | 서울시 종로구 대학로 12길 38 (정보빌딩)
전 화 | (02)3673-0037(편집부) / (02)3673-0114(代)
팩 스 | (02)3673-0260
등 록 | 1990년 2월 14일 제1-1013호
홈페이지 | www.infopub.co.kr

I S B N | 978-89-5674-734-7

이 책은 저작권법에 따라 보호받는 저작물이므로 무단 전재와
무단 복제를 금하며, 이 책 내용의 전부 또는 일부를 사용하려면 반드시
저작권자와 정보문화사 발행인의 서면동의를 받아야 합니다.

※ 책값은 뒤표지에 있습니다.
※ 잘못된 책은 구입한 서점에서 바꿔 드립니다.

머리말

이 세상에는 컴퓨터 프로그램 또는 소프트웨어를 만들기 위한 다양한 컴퓨터 프로그래밍 언어가 존재합니다. 컴퓨터 프로그래밍 언어는 컴퓨터의 연산능력을 활용하여 특정 목적을 달성하는 코드를 작성하는 데 사용되는 언어입니다. 자바, C, C++, C#, 파이썬 등은 모두 컴퓨터 프로그래밍 언어입니다.

컴퓨터 프로그래밍 언어를 활용해서 컴퓨터 프로그램이나 소프트웨어를 만드는 사람을 ‘프로그래머’라고 합니다. 프로그래머는 자신이 가장 익숙하고 잘 활용할 수 있는 프로그래밍 언어를 가지고 소프트웨어를 개발합니다. 하지만 프로그램이 달성하고자 하는 목적이나 성격, 구동되는 환경에 따라 최적인 프로그래밍 언어가 있을 수 있습니다. 프로그래머는 소프트웨어 개발에 소요되는 시간이나 비용을 고려해야 하며, 개발하고자 하는 소프트웨어의 성격, 예컨대 빠른 시간에 결과를 알고 싶은 것인지, 많은 연산을 수행하는 것인지, 또는 병렬 처리가 필요한 것인지 등에 따라 적합한 프로그래밍 언어를 결정해야 합니다. 따라서 프로그래머에게 있어서 자신이 만들고자 하는 소프트웨어에 가장 적합한 프로그래밍 언어를 선택하는 것은 중요한 일입니다.

이 책에서 다루게 될 파이썬은 프로그래밍에 드는 노력 대비 가성비로 따져볼 때 매우 훌륭한 프로그래밍 언어입니다.

파이썬은 네덜란드 출신의 프로그래머인 귀도 반 로섬(Guido van Rossum)이 1991년 개발한 스크립트 언어입니다. 스크립트 언어는 프로그래밍이 직관적이고 배우기 쉽다는 것이 장점이며, 컴파일(Compile)이라는 과정 없이 인터프리터(Interpreter, 해석기)가 코드를 한 줄씩 바로 해석하여 결과를 보여주기 때문에 프로그래머와 대화하듯이 프로그래밍이 가능합니다.

파이썬은 스크립트 언어이지만 매우 광범위하고 다양한 종류의 라이브러리가 있어 C/C++, Java 등의 언어로 구현할 수 있는 대부분의 프로그램을 작성할 수 있습니다. 또한, 동일한 목적의 프로그램을 C로 작성할 때와 비교해 볼 때, 파이썬으로 작성하게 되면 절반도 되지 않는 노력으로 프로그램을 만들 수 있습니다. 파이썬은 데이터 처리에 관련된 다양한 확장 라이브러리들이 많아 최근 트렌드가 되고 있는 머신러닝이나 AI 분야에서 사용되는 가장 인기

있는 프로그래밍 언어이기도 합니다.

그렇다고 파이썬이 모든 것을 해결해주는 만병통치, 팔방미인이 되는 프로그래밍 언어는 아닙니다. 예를 들어, CPU 연산을 많이 필요로 하는 수학 계산 및 그래픽 처리나, 멀티태스킹이 많이 필요한 병렬처리 프로그램, I/O가 매우 빈번하게 발생하는 프로그램은 C나 C++ 같은 저수준의 프로그래밍 언어로 개발하는 것이 좋습니다.

이 책은 파이썬을 배우기 위해 입문하는 초보자를 위해 쓰였으며, 파이썬에 익숙한 분들도 필요할 때마다 사전식으로 쉽게 찾아서 참고할 수 있도록 구성했습니다.

책은 총 5개 파트, 200개의 예제로 구성되어 있습니다. 처음부터 차근차근 예제를 통해 파이썬 기초를 다지고 간단한 프로그램을 만들 수 있는 실력까지 향상될 수 있도록 했습니다.

1파트 ‘Python 프로그래밍 첫발 내딛기’는 15개 예제로 구성되어 있으며, 파이썬 프로그래밍을 위해 알아야 할 가장 기초적인 개념에 대해 배웁니다.

2파트 ‘Python 프로그래밍 기초 다지기’는 44개 예제로 구성되어 있으며, 파이썬의 자료형과 함수, 클래스, 예외처리와 같은 파이썬의 기본적인 자료구조와 문법에 대해 배웁니다.

3파트 ‘Python 프로그래밍 실력 다지기’는 77개 예제로 구성되어 있으며, 간단한 예제를 통해 파이썬의 자료형과 내장함수 등에 대한 활용 방법과 이해를 돋도록 했습니다.

4파트 ‘Python 프로그래밍 응용 다지기’는 44개 예제로 구성되어 있으며, 파일 입출력 및 파일처리, 시간 관련 응용, 문자열 파싱 및 처리, URL을 활용한 인터넷 접속 등과 같은 간단한 응용 예제를 통해 파이썬 프로그래밍 응용 능력을 향상하도록 했습니다.

5파트 ‘Python으로 간단한 프로그램 작성하기’는 20개 예제로 구성되어 있으며 데이터 처리, 웹서버 로그 처리, 간단한 슈팅 게임 만들기, 클라이언트 서버 프로그램 작성하기, 메신저 프로그램 만들기 등과 같은 프로그램을 구현해봄으로써 실무에 응용할 수 있는 능력을 키우도록 했습니다.

파이썬에 막 입문하는 분들은 이 책의 처음부터 정독하며 예제를 하나하나 손으로 입력하면서 학습하기 바랍니다. 책에 수록된 소스 코드와 데이터 파일은 정보문화사 홈페이지(<http://www.infopub.co.kr>)의 자료실이나 저자 블로그(<http://blog.naver.com/samsjang>)에서 다운로드 받을 수 있습니다.

책 한권으로 파이썬의 대가가 될 수는 없겠지만, 파이썬을 전혀 모르는 이도 이 책을 통해 파이썬이라는 언어에 익숙해지고 잘 활용할 수 있는 프로그래밍 언어가 될 수 있기 바랍니다.

마지막으로 이 책을 쓰면서 도움을 받은 분들에게 감사의 말씀을 전합니다. 이 책이 출판될 수 있도록 많은 도움을 주신 정보문화사 관계자분들께 감사의 말씀을 드립니다.

그리고 현재 병마와 싸우고 계신 부평 어머니의 쾌차를 바랍니다. 끝으로 저를 낳아주신 진주 어머니 항상 건강하시기를 기원합니다.

저자 장삼용

차례

머리말	003
이 책의 구성	014
파이썬 개발 환경 준비하기	016

PART 1 입문

Python 프로그래밍 첫발 내딛기

001 대화식 모드로 프로그래밍하기	024
002 텍스트 에디터로 프로그래밍하기	026
003 변수명 만들기	028
004 변수에 값 대입하기	030
005 주석 처리하기(#)	031
006 자료형 개념 배우기	032
007 자료형 출력 개념 배우기(print)	034
008 들여쓰기 개념 배우기	036
009 if문 개념 배우기 ①(if~else)	038
010 if문 개념 배우기 ②(if~elif)	040
011 for문 개념 배우기 ①(for)	041
012 for문 개념 배우기 ②(for~continue~break)	043
013 for문 개념 배우기 ③(for~else)	045
014 while문 개념 배우기(while~continue~break)	046
015 None 개념 배우기	049

016	정수형 자료 이해하기	052
017	실수형 자료 이해하기	054
018	복소수형 자료 이해하기	056
019	대입 연산자 이해하기(=)	057
020	사칙 연산자 이해하기(+, -, *, /, **)	058
021	연산자 축약 이해하기(+=, -=, *=, /=)	060
022	True와 False 이해하기	061
023	관계 연산자 이해하기(==, !=, <, <=, >, >=)	062
024	논리 연산자 이해하기(and, or, not)	064
025	비트 연산자 이해하기(&, , ~, ^, >>, <<)	066
026	시퀀스 자료형 이해하기	069
027	시퀀스 자료 인덱싱 이해하기	071
028	시퀀스 자료 슬라이싱 이해하기	073
029	시퀀스 자료 연결 이해하기(+)	075
030	시퀀스 자료 반복 이해하기(*)	076
031	시퀀스 자료 크기 이해하기(len)	077
032	멤버체크 이해하기(in)	078
033	문자열 이해하기	079
034	문자열 포맷팅 이해하기	080
035	이스케이프 문자 이해하기	082
036	리스트 이해하기([])	084
037	튜플 이해하기(())	085
038	사전 이해하기({ })	086
039	함수 이해하기(def)	087
040	함수 인자 이해하기	090
041	지역변수와 전역변수 이해하기(global)	093
042	함수 리턴값 이해하기(return)	095

차례

043	파이썬 모듈 이해하기	096
044	파이썬 패키지 이해하기	098
045	파이썬 모듈 임포트 이해하기 ① (import)	099
046	파이썬 모듈 임포트 이해하기 ② (from~import)	101
047	파이썬 모듈 임포트 이해하기 ③ (import~as)	103
048	파일 열고 닫기(open, close)	104
049	클래스 이해하기(class)	106
050	클래스 멤버와 인스턴스 멤버 이해하기	108
051	클래스 메소드 이해하기	110
052	클래스 생성자 이해하기	111
053	클래스 소멸자 이해하기	113
054	클래스 상속 이해하기	114
055	예외처리 이해하기 ① (try~except)	116
056	예외처리 이해하기 ② (try~except~else)	118
057	예외처리 이해하기 ③ (try~except~finally)	119
058	예외처리 이해하기 ④ (try~except Exception as e)	120
059	예외처리 이해하기 ⑤ (try~except 특정 예외)	121

PART 3 중급

Python 프로그래밍 실력 다지기

060	사용자 입력받기(input)	124
061	자료형 확인하기(type)	125
062	나눗셈에서 나머지만 구하기(%)	127
063	몫과 나머지 구하기(divmod)	128
064	10진수를 16진수로 변환하기(hex)	129
065	10진수를 2진수로 변환하기(bin)	131
066	2진수, 16진수를 10진수로 변환하기(int)	132
067	절대값 구하기(abs)	133

068	반올림수 구하기(round)	134
069	실수형 자료를 정수형 자료로 변환하기(int)	135
070	정수형 자료를 실수형 자료로 변환하기(float)	136
071	정수 리스트에서 소수만 걸러내기(filter)	137
072	최대값, 최소값 구하기(max, min)	139
073	1바이트에서 하위 4비트 추출하기	141
074	1바이트에서 상위 4비트 추출하기	142
075	문자열에서 특정 위치의 문자 얻기	143
076	문자열에서 지정한 구간의 문자열 얻기	144
077	문자열에서 홀수 번째 문자만 추출하기	146
078	문자열을 거꾸로 만들기	147
079	두 개의 문자열 합치기(+)	148
080	문자열을 반복해서 새로운 문자열로 만들기(*)	149
081	문자열에서 특정 문자가 있는지 확인하기(in)	150
082	문자열에서 특정 문자열이 있는지 확인하기(in)	151
083	문자열 길이 구하기(len)	152
084	문자열이 알파벳인지 검사하기(isalpha)	154
085	문자열이 숫자인지 검사하기(isdigit)	155
086	문자열이 알파벳 또는 숫자인지 검사하기(isalnum)	156
087	문자열에서 대소문자 변환하기(upper, lower)	157
088	문자열에서 좌우 공백 제거하기(lstrip, rstrip, strip)	158
089	문자열을 수치형 자료로 변환하기(int, float)	159
090	수치형 자료를 문자열로 변환하기(str)	161
091	문자열에 있는 문자(열) 개수 구하기(count)	162
092	문자열에서 특정 문자(열) 위치 찾기(find)	163
093	문자열을 특정 문자(열)로 분리하기(split)	164
094	문자열을 특정 문자(열)로 결합하기(join)	166
095	문자열에서 특정 문자(열)를 다른 문자(열)로 바꾸기(replace)	167
096	문자열을 바이트 객체로 바꾸기(encode)	168
097	바이트 객체를 문자열로 바꾸기(decode)	170
098	문자열을 정렬하기(sorted, join)	171

차례

099	순차적인 정수 리스트 만들기(range)	173
100	리스트에서 특정 위치의 요소 얻기	174
101	리스트에서 특정 요소의 위치 구하기(index)	175
102	리스트에서 특정 위치의 요소를 변경하기	177
103	리스트에서 특정 구간에 있는 요소 추출하기	178
104	리스트에서 짝수 번째 요소만 추출하기	179
105	리스트 요소 순서를 역순으로 만들기 ① (reverse)	180
106	리스트 요소 순서를 역순으로 만들기 ② (reversed)	181
107	리스트 합치기(+)	182
108	리스트 반복하기(*)	183
109	리스트에 요소 추가하기(append)	184
110	리스트의 특정 위치에 요소 삽입하기(insert)	185
111	리스트의 특정 위치의 요소 제거하기(del)	186
112	리스트에서 특정 요소 제거하기(remove)	187
113	리스트에서 특정 구간에 있는 모든 요소 제거하기	188
114	리스트에 있는 요소 개수 구하기(len)	189
115	리스트에서 특정 요소 개수 구하기(count)	190
116	리스트 제거하기(del)	191
117	리스트 요소 정렬하기 ① (sort)	192
118	리스트 요소 정렬하기 ② (sorted)	193
119	리스트 요소 무작위로 섞기(shuffle)	194
120	리스트의 모든 요소를 인덱스와 쌍으로 추출하기(enumerate)	195
121	리스트의 모든 요소의 합 구하기(sum)	197
122	리스트 요소가 모두 참인지 확인하기(all, any)	198
123	사전에 요소 추가하기	200
124	사전의 특정 요소값 변경하기	202
125	사전의 특정 요소 제거하기(del)	203
126	사전의 모든 요소 제거하기(clear)	204
127	사전에서 키만 추출하기(keys)	205
128	사전에서 값만 추출하기(values)	207
129	사전 요소를 모두 추출하기(items)	208

130	사전에 특정 키가 존재하는지 확인하기(<code>in</code>)	210
131	사전 정렬하기(<code>sorted</code>)	212
132	문자 코드값 구하기(<code>ord</code>)	215
133	코드값에 대응하는 문자 얻기(<code>chr</code>)	217
134	문자열로 된 식을 실행하기(<code>eval</code>)	218
135	이름없는 한줄짜리 함수 만들기(<code>lambda</code>)	219
136	인자를 바꾸어 함수를 반복 호출하여 결과값 얻기(<code>map</code>)	221

PART 4

활용

Python 프로그래밍 응용 다지기

137	텍스트 파일을 읽고 출력하기(<code>read</code>)	226
138	텍스트 파일을 한줄씩 읽고 출력하기 ① (<code>readline</code>)	228
139	텍스트 파일을 한줄씩 읽고 출력하기 ② (<code>readlines</code>)	230
140	화면에서 사용자 입력을 받고 파일로 쓰기(<code>write</code>)	231
141	텍스트 파일에 한줄씩 쓰기(<code>writelines</code>)	232
142	텍스트 파일 복사하기(<code>read, write</code>)	234
143	바이너리 파일 복사하기(<code>read, write</code>)	235
144	파일을 열고 자동으로 닫기(<code>with~as</code>)	237
145	파일의 특정 부분만 복사하기(<code>seek, read, write</code>)	238
146	파일 크기 구하기(<code>os.path.getsize</code>)	240
147	파일 삭제하기(<code>os.remove</code>)	241
148	파일이름 바꾸기(<code>os.rename</code>)	242
149	파일을 다른 디렉터리로 이동하기(<code>os.rename</code>)	243
150	디렉터리에 있는 파일목록 얻기(<code>os.listdir, glob.glob</code>)	245
151	현재 디렉터리 확인하고 바꾸기(<code>os.getcwd, os.chdir</code>)	247
152	디렉터리 생성하기(<code>os.mkdir</code>)	248
153	디렉터리 제거하기(<code>os.rmdir</code>)	249
154	하위 디렉터리 및 파일 전체 삭제하기(<code>shutil.rmtree</code>)	250
155	파일이 존재하는지 체크하기(<code>os.path.exists</code>)	251

차례

156	파일인지 디렉터리인지 확인하기(<code>os.path.isfile</code> , <code>os.path.isdir</code>)	252
157	현재 시간을 년-월-일 시:분:초로 출력하기(<code>localtime</code> , <code>strftime</code>)	253
158	올해 경과된 날짜수 계산하기(<code>localtime</code>)	256
159	오늘의 요일 계산하기(<code>localtime</code>)	257
160	프로그램 실행 시간 계산하기(<code>datetime.now</code>)	258
161	주어진 숫자를 천 단위로 구분하기	260
162	문자열의 각 문자를 그 다음 문자로 변경하기	262
163	URL에서 호스트 도메인 추출하기	263
164	URL에서 쿼리 문자열 추출하기	264
165	스택 구현하기(<code>append</code> , <code>pop</code>)	266
166	문장에 나타나는 문자 빈도수 계산하기	268
167	텍스트 파일에 있는 단어 개수 계산하기	270
168	파일에서 특정 단어 개수 계산하기	271
169	파일에서 특정 문자열 교체하기	273
170	URL에 접속하여 HTML 페이지 화면에 출력하기	274
171	URL에 접속하여 HTML 페이지를 파일로 저장하기	276
172	인터넷에 있는 이미지를 내 PC로 저장하기	278
173	인터넷에 있는 대용량 파일을 내 PC로 저장하기	280
174	10MB 파일을 1MB 파일 10개로 분리하기	282
175	1MB 파일 10개를 합쳐서 10MB 파일로 만들기	284
176	파일을 ZIP 압축 파일로 만들기	286
177	디렉터리를 하나의 ZIP 압축 파일로 만들기	288
178	ZIP 파일 압축 풀기	290
179	로또 번호 추출기 만들기	291
180	남녀 파트너 정해주기 프로그램 만들기(<code>zip</code>)	293

PART 5**실무****Python으로 간단한 프로그램 작성하기**

181	데이터 처리하기 ① 연도별 출생아 수 계산하기	298
182	데이터 처리하기 ② 연도별 성별 출생아 수 계산하기	302
183	데이터 처리하기 ③ 연도별 인기있는 상위 10개 성별 출생아 이름 구하기	305
184	웹서버 로그 처리하기 ① 총 페이지뷰 수 계산하기	308
185	웹서버 로그 처리하기 ② 고유 방문자 수 계산하기	311
186	웹서버 로그 처리하기 ③ 총 서비스 용량 계산하기	313
187	웹서버 로그 처리하기 ④ 사용자별 서비스 용량 계산하기	315
188	간단한 슈팅게임 만들기 ① 게임화면 구성하기	317
189	간단한 슈팅게임 만들기 ② 전투기 배치하기	322
190	간단한 슈팅게임 만들기 ③ 적 날아오게 하기	327
191	간단한 슈팅게임 만들기 ④ 무기 발사하기	331
192	간단한 슈팅게임 만들기 ⑤ 게임규칙 적용하기	336
193	에코 서버 만들기 ①	343
194	에코 클라이언트 만들기 ①	347
195	에코 서버 만들기 ②	351
196	에코 클라이언트 만들기 ②	354
197	파일 송신 프로그램 만들기	356
198	파일 수신 프로그램 만들기	359
199	채팅 서버 만들기	362
200	채팅 클라이언트 만들기	369
	찾아보기	374

이 책의 구성

① 예제 제목

해당 예제의 번호와 제목을 가장 핵심적인 내용으로 나타냅니다.

② 학습 내용

해당 예제에서 배울 내용을 핵심적으로 나타냅니다.

③ 힌트 내용

예제에 대한 힌트나 시간을 절약 할 수 있는 방법, 앞에서 설명한 내용과 관련된 또 다른 과정, 일반적으로 알려진 기본 방법 이외에 숨겨진 기능을 설명해 줍니다.

④ 소스

예제의 파일명을 나타냅니다. 예제 파일은 정보문화사 홈페이지(www.infopub.co.kr)의 자료실에서 다운로드 받을 수 있습니다.

⑤ 예제 소스

해당 단락에서 배울 내용의 전체 예제(소스)를 나타냅니다.

⑥ 줄 번호

해당 예제에서 설명하는 기본적인 용어에 대한 정의를 명쾌하게 내려줍니다.

1 사전에 특정 키가 존재하는지 확인하기(in)

- ② 학습 내용 : 특정 값이 사전의 키로 존재하는지 확인하는 방법에 대해 배웁니다.
③ 힌트 내용 : in 키워드를 이용하여 특정 값이 사전의 키로 존재하는지 확인할 수 있습니다.

4 소스 : 130.py

```
1: names = {'Mary':10999, 'Sams':2111, 'Aimy':9778, 'Tom':20245,  
2:           'Michale':27115, 'Bob':5887, 'Kelly':7855}  
3: k = input('이름을 입력하세요: ')  
4: if k in names:  
5:     print('이름이 <%s>인 출생아 수는 <%d>명입니다.' %(k, names[k]))  
6: else:  
7:     print('자료에 <%s>인 이름이 존재하지 않습니다.' %k)
```

사전 dict에 k가 키로 있는지 없는지 확인하는 방법은 다음과 같습니다.

k in dict

만약 k가 dict에 키로 존재하면 True, 존재하지 않으면 False입니다.

예제는 사용자로부터 이름을 입력받고 이에 해당하는 이름의 출생아 수를 사전자료에서 추출하여 결과를 화면에 출력하는 코드입니다.

- 1-2 ◆ 이름이 키이고 출생아 수가 값인 사전자료 names를 정의합니다.

- 3 ◆ 사용자로부터 이름을 입력받고 그 값을 변수 k에 대입합니다.

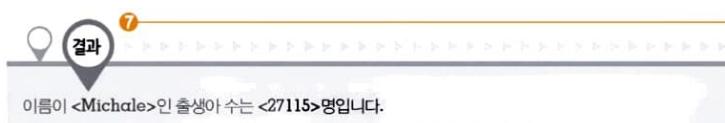
- 4-5 ◆ k가 사전 자료 names의 키로 존재하면, 키와 이 키에 대응하는 값을 출력합니다.

- 6-7 ◆ k가 사전 자료 names의 키로 존재하지 않으면 사용자가 입력한 값이 names의 키로 존재하지 않는다는 메시지를 출력합니다.

이 책은 여러분이 파이썬을 배우는 데 집중할 수 있도록 다음과 같은 몇 가지 특징으로 구성되어 있습니다.

Part 3 > Python 프로그래밍 실력 다지기

코드를 실행하고 사용자가 'Michale'이라는 이름을 입력하면 다음과 같은 메시지가 출력됩니다.



결과

7

이름이 <Michale>인 출생아 수는 <27115>명입니다.



NOTE | 사진에 특정 값이 존재하는지 확인하는 방법 | 8

사진에 특정 값이 키에 대응되는 값으로 존재하는지 확인하는 방법은 앞서 배운 `values()`를 이용하는 것입니다.

다음은 2111이 사진자료에 값으로 존재하는지 확인하는 코드입니다.

```
names = {'Mary':10999, 'Sams':2111, 'Aimy':9778, 'Tom':20245,  
        'Michale':27115, 'Bob':5887, 'Kelly':7855}
```

```
if 2111 in names.values():  
    print('주어진 값이 사진에 존재합니다.')  
else:  
    print('주어진 값이 사진에 존재하지 않습니다.)
```

7 결과 화면

설명한 예제의 입력, 컴파일, 링크 과정을 거쳐 예제의 실행 결과를 보여줍니다. 이 결과와 다르게 나온다면 다시 한번 확인해 보는 것이 좋습니다.

8 NOTE

예제를 학습해 보면서 현재 내용과 관련된 추가 정보나 주의할 점, 초보자가 종종 놓칠 수 있는 내용들을 알려줍니다.

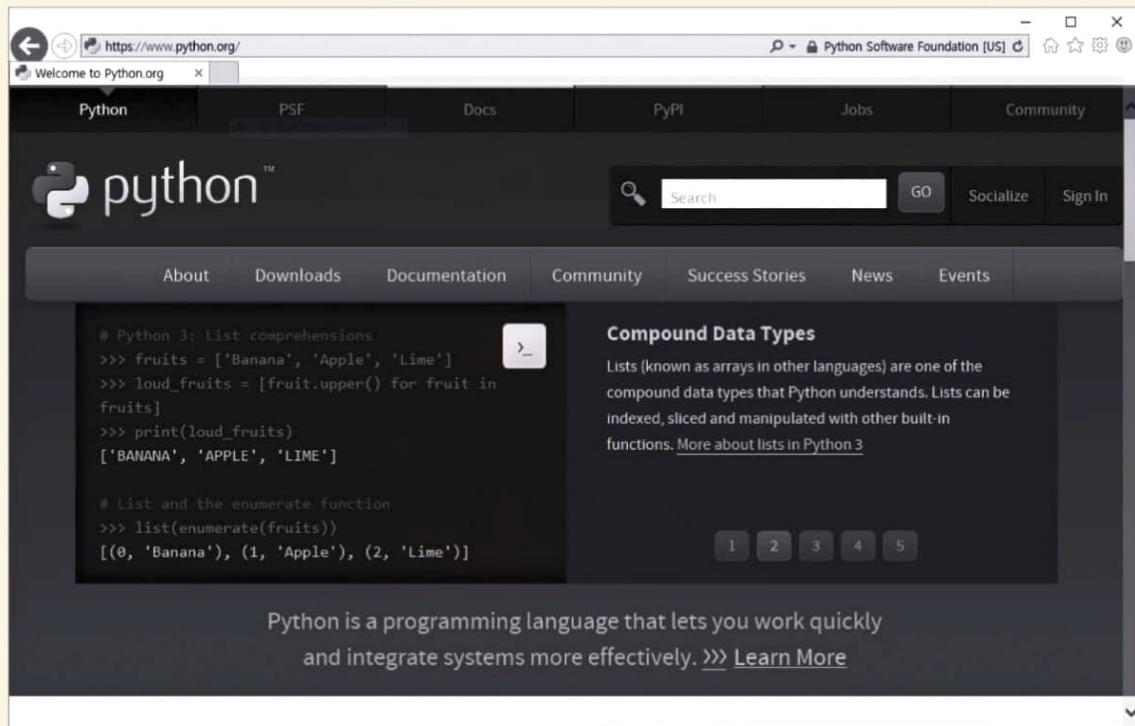
파이썬 개발 환경 준비하기

파이썬을 이용해 프로그래밍 입문을 하기 위해서 가장 먼저 해야 할 일은 여러분의 컴퓨터에 파이썬 개발 환경을 갖추는 일입니다. 이 책에서 다루는 모든 코드는 파이썬 3.5.x 버전으로 작성되어 있습니다.

파이썬 3.5.x 또는 3.6.x 버전 설치하기

파이썬으로 프로그램을 개발하기 위해서는 파이썬 인터프리터가 필요합니다. 파이썬 인터프리터는 우리가 작성한 파이썬 코드를 해석하여 처리하고 그 결과를 제시합니다.

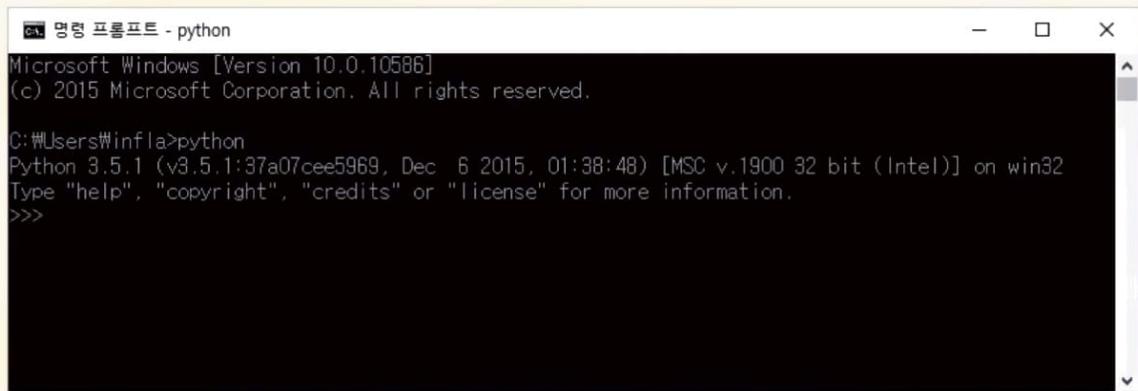
앞으로 파이썬 인터프리터를 간단히 파이썬으로 부르도록 하겠습니다. 파이썬 공식 홈페이지는 <https://www.python.org>입니다. 파이썬 공식 홈페이지에는 파이썬과 관련한 매우 방대한 자료들을 제공하고 있습니다.



| 그림 1 | 파이썬 공식 홈페이지

파이썬은 대부분의 리눅스 계열 OS에 기본적으로 탑재되어 있으며, 최근에는 Mac OS X나 윈도우에도 파이썬이 기본적으로 탑재되어 있습니다. 윈도우의 명령 프롬프트나 Mac OS X의 터미널 프로그램을 열고 python을 입력하여 파이썬이 구동되는지 확인합니다.

여러분의 컴퓨터에 파이썬이 설치되어 있다면 그림 2에서 보이는 것처럼 파이썬이 실행되면 Python 3.5.1과 같이 버전 정보를 출력합니다. 만약 여러분의 컴퓨터에 설치된 파이썬 버전이 3.5.x 또는 3.6.x 버전이 아니라면 다음과 같이 여러분의 OS에 맞는 파이썬 최신 버전을 설치합니다. 이후 설치는 파이썬 3.6.x 버전 기준으로 설명합니다.



```
명령 프롬프트 - python
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

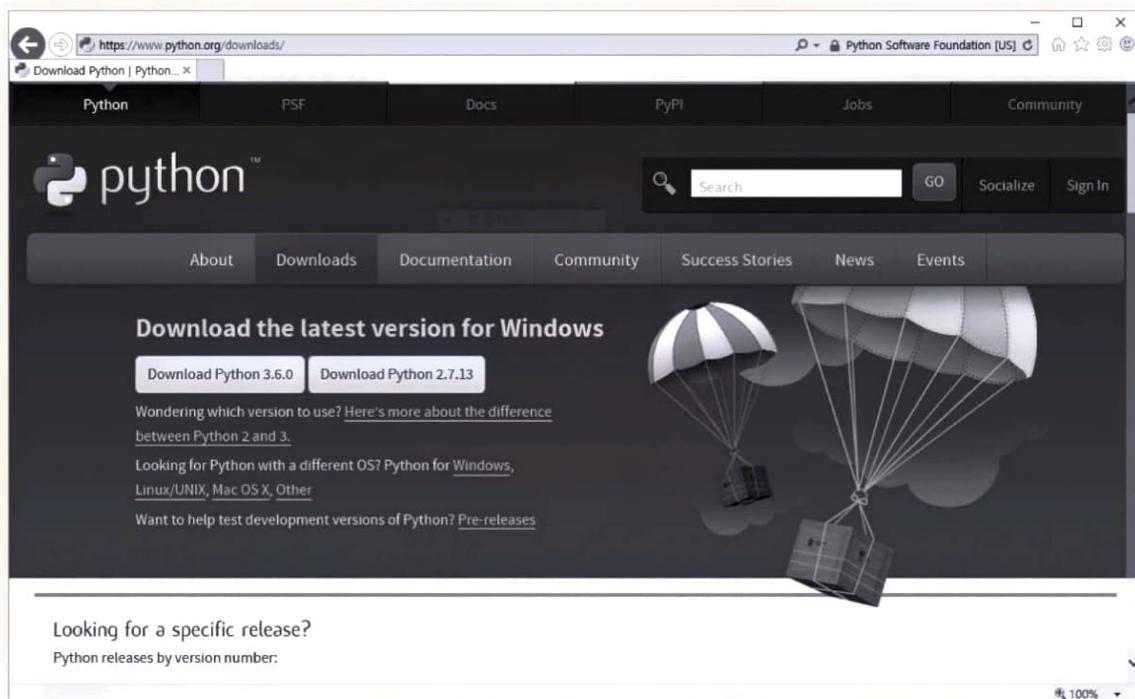
C:\Users\Winfla>python
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  6 2015, 01:38:48) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

| 그림 2 | 윈도우 명령 프롬프트에서 파이썬 실행 화면

윈도우용 파이썬 설치

1. 파이썬 바이너리 다운로드

파이썬 공식 홈페이지의 상단 메뉴를 보면 **downloads**라는 이름의 탭이 있습니다. 이 탭을 누르면 여러분의 OS에 맞는 파이썬을 다운로드 받을 수 있습니다. 여러분의 OS가 윈도우인 경우 다음과 같은 화면을 볼 수 있습니다. 여기서 **Download Python 3.6.x** 탭을 누르고 파이썬 바이너리 파일을 다운로드 받습니다. 참고로 파이썬 3.5.x 버전 이상은 윈도우 7 이상에서 구동됩니다.

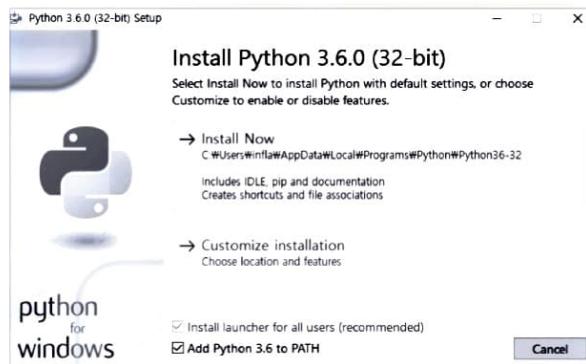


| 그림 3 | 파이썬 바이너리 다운로드 페이지

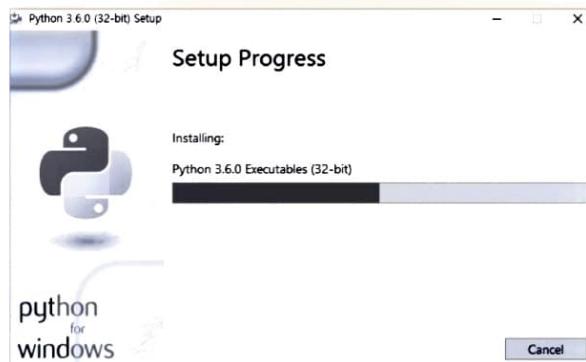
2. 파이썬 설치

다운로드 받은 파이썬 실행 파일은 `python-3.6.x.exe`와 같은 형식의 이름으로 되어 있습니다. 이 파일을 실행하고 다음과 같은 순서로 설치합니다.

Add Python 3.6.x to PATH에 체크를 하고
Install Now를 클릭합니다.



파이썬 3.6.x와 필요한 모듈들을 자동으로
알아서 설치해 줍니다.



설치가 성공적으로 완료되면 다음과 같은
화면이 나타납니다. Close 버튼을 눌러 설치
를 종료합니다.



3. 파이썬 설치 확인

윈도우 명령 프롬프트를 열고 명령 프롬프트에서 `python`을 입력하고 `Enter`를 눌렀을 때 그림 2와 같은 화면이 나와야 제대로 설치된 것입니다.

OS X용 파이썬 설치

윈도우용 파이썬 설치와 비슷합니다.

1. 파이썬 OS X용 바이너리 다운로드

파이썬 공식 홈페이지 [downloads](#) 템을 누르고 MAC OS X 메뉴에서 OS X용 파이썬 3.5.x 바이너리 파일을 다운로드 받습니다.

2. 파이썬 설치

다운로드 받은 파이썬 실행 파일은 `python-3.6.x-macosx10.6.pkg` 같은 형식의 이름으로 되어 있습니다. 이 파일을 실행하고 디폴트 옵션으로 설치합니다.

리눅스와 우분투(Ubuntu)용 파이썬 3.5.x 설치

리눅스 계열의 OS에는 파이썬이 기본적으로 탑재되어 있습니다. 리눅스 셸에서 `python`을 입력하여 파이썬을 실행합니다. 설치된 파이썬 버전이 2.7.x 버전이면 다음의 명령을 통해 파이썬 3.5.x 또는 3.6.x를 다운로드 받아 설치합니다.

리눅스

```
$ yum search python3    # yum으로 파이썬 3을 검색함  
$ sudo yum install <검색된 파이썬 패키지 이름>
```

우분투

```
$ sudo apt-get install python3.5.x
```

리눅스와 우분투에는 파이썬 프로그래밍을 위한 기본 에디터인 IDLE이 설치되어 있지 않기 때문에 아래의 명령으로 IDLE 프로그램을 다운로드 받아 설치합니다.

리눅스

```
$ yum search idle      # yum으로 IDLE을 검색함  
$ sudo yum install <검색된 IDLE 패키지>
```

우분투

```
$ sudo apt-get install idle3
```

파이썬을 성공적으로 설치하였으면 이제 모든 준비가 끝났습니다.

1

P A R T

입문

Python 프로그래밍 첫발 내딛기

입문

001

대화식 모드로 프로그래밍하기

- 학습 내용 : 대화식 프로그래밍의 개념을 이해하고 대화식 모드로 프로그래밍하는 방법을 배웁니다.
- 힌트 내용 : '>>>'는 파이썬이 대화식 모드임을 나타내는 프롬프트입니다.

```
>>> print('안녕하세요')
```

안녕하세요

파이썬은 인터프리터 언어이기 때문에 소스 코드 한 라인씩 순서대로 실행하고 그 결과를 출력합니다. 이와 같이 프로그래머가 한 라인의 소스 코드를 입력하면 인터프리터가 곧바로 해석하여 그 결과를 프로그래머에 제시하는 방식을 대화식 프로그래밍이라 부릅니다.

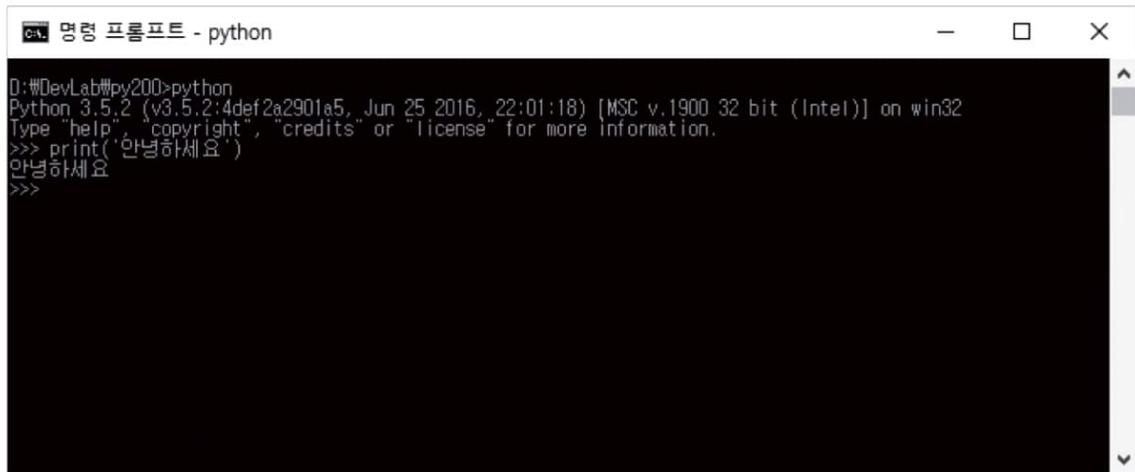
대화식 모드로 파이썬 프로그래밍을 하는 방법은 대표적으로 두 가지가 있습니다.

- 원도우 명령 프롬프트를 열고 파이썬을 실행한다.
- IDLE을 실행한다.

두 가지 방법 모두 실행하면 화면에 보이는 '>>>'은 파이썬 인터프리터의 프롬프트입니다.

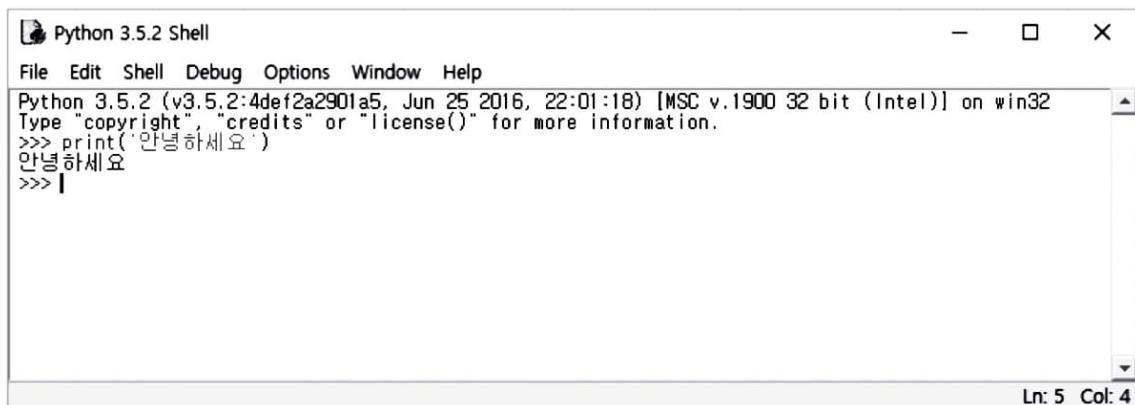
IDLE은 파이썬을 설치하면 기본적으로 같이 설치되는 파이썬 프로그래밍을 위한 단순한 개발환경입니다. IDLE을 실행하려면 원도우 명령프롬프트나 원도우 하단 작업표시줄 왼쪽에 있는 원도우 검색창에서 IDLE을 입력하면 됩니다.

'>>>' 부분에 파이썬 코드를 한 라인 작성하고 **Enter**를 누르면 파이썬 인터프리터는 곧바로 이를 해석하여 결과를 보여줍니다. 이 책에서 '>>>'가 표시된 부분은 모두 대화식 모드로 프로그래밍하는 것으로 생각하면 됩니다.



```
D:\#DevLab#\py200>python
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('안녕하세요')
안녕하세요
>>>
```

| 그림 001-1 | 원도우 명령 프롬프트에서 대화식 모드로 파이썬 프로그래밍하는 방법



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print('안녕하세요')
안녕하세요
>>> |
```

| 그림 001-2 | IDLE을 실행하여 대화식 모드로 파이썬 프로그래밍하는 방법

입문

002

텍스트 에디터로 프로그래밍하기



- **학습 내용 :** 에디터를 이용하여 파이썬 프로그래밍을 하는 방법에 대해 배웁니다.
- **힌트 내용 :** IDLE에는 텍스트 에디터가 탑재되어 있습니다.

소스 : 002.py

```
1: print('안녕하세요')
2: a = 1
3: b = 1
4: print(a+b)
```

코드 왼쪽에
표시된 숫자는
라인을 나타내는
것으로 실제
코드에서는
입력하지
않습니다.

라인 단위로 입력하고 실행하는 인터페이스는 코드 한 라인의 실행 결과를 곧바로 확인하는 용도로는 유용하지만, 코드가 복잡해지고 길어지면 다소 효율성이 떨어지게 마련입니다.

IDLE을 실행하고 상단 메뉴에서 [File] – [New File]을 클릭해 텍스트 에디터를 열어봅니다. 텍스트 에디터에 위 코드를 입력합니다. 입력 후 파일로 저장하기 위해 **Ctrl + S**를 누르고 ‘002.py’로 저장합니다. **F5**를 눌러 저장된 코드를 실행해봅니다.

The screenshot shows the IDLE Python editor window. The title bar says '002.py - D:/작업실/도서저작실/파이썬200제/소스코드/002.py (3.5.2)'. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code area contains the following Python code:

```
print('안녕하세요')
a = 1
b = 1
print(a+b)
```

The status bar at the bottom right shows 'Ln: 5 Col: 0'.

| 그림 002-1 | IDLE 텍스트 에디터로 파이썬 프로그래밍하는 방법

```

Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART: D:/작업실/도서저작실/파이썬200제/소스코드/002.py =====
안녕하세요
2
>>> |
Ln: 7 Col: 4

```

그림 002-2 | 실행 결과 화면

이 방법은 파일에 작성된 파이썬 코드를 한 번에 일괄적으로 실행하는 배치(batch) 형식으로 동작합니다. 이는 프로그래머가 일반적으로 코딩하는 방식입니다.

텍스트 에디터에서 무언가를 수정한 후 F5로 코드를 실행하면 코드가 수정되었으니 저장할 것인지 묻는 창이 나타납니다. 저장하지 않으면 실행되지 않으니 [yes]를 눌러 저장하고 실행하면 됩니다.

윈도우에서 기본적으로 제공하는 메모장과 같이 일반 텍스트 에디터를 이용해서 프로그래밍할 수도 있습니다. 윈도우 메모장을 열고 위 코드를 입력한 후 '002.py'로 저장합니다. 윈도우 명령 프롬프트를 열고 002.py가 저장된 폴더로 이동합니다. 명령 프롬프트에서 python 002.py로 실행하여 결과를 확인합니다.

```

명령 프롬프트
D:\#DevLab#\py200>python 002.py
안녕하세요
2
D:\#DevLab#\py200>

```

그림 002-3 | 원도우 명령 프롬프트에서 파이썬 프로그램 실행하는 방법

이 외에 파이썬을 위한 무료로 사용할 수 있는 에디터로 PyCharm, IPython notebook, SciTE 에디터 등이 있습니다. 이 책에서 다루는 대부분의 코드는 IDLE에서 제공하는 텍스트 에디터로도 충분하므로 이를 활용하도록 합니다.

이 책에서 라인 번호가 적혀 있는 예는 IDLE의 텍스트 에디터를 이용해 프로그래밍한 것으로 생각하면 됩니다.

입문

003

변수명 만들기



- **학습 내용 :** 어떤 값을 임시로 저장하는 변수의 이름을 만드는 방법과 규칙에 대해 배웁니다.
- **힌트 내용 :** 수학 방정식 $x + y = 1$ 에서 x, y 를 변수라 합니다.

```
>>> _myname = 'samsjang'  
>>> my_name = '홍길동'  
>>> MyName2 = 'Hong gil-dong'  
>>> counter = 1  
>>> Counter = 2
```

파이썬의 변수명 규칙은 C나 C++ 등 다른 언어에서 사용되는 변수명 규칙과 비슷합니다.

변수명의 첫 문자는 밑줄 문자(언더스코어) ‘_’ 또는 영문자로 시작해야 합니다. 따라서 다음과 같은 변수명은 잘못된 예입니다.

l_unit, %var, @address

변수명의 두 번째 문자부터는 알파벳, 숫자 그리고 밑줄 문자를 사용할 수 있습니다. 변수명은 대소 문자를 구분하므로 `counter`와 `Counter`는 다른 변수가 됩니다.

하지만 파이썬에서 이미 사용하고 있는 일부 단어는 변수명으로 사용할 수 없습니다. 이러한 단어를 **파이썬 예약어**라고 합니다.

파이썬 예약어에는 제어문으로 사용되는 `if`, `elif`, `while`, `for`가 있고 함수를 정의할 때 사용되는 `def`, 클래스를 나타내는 `class` 등 많은 예약어들이 있습니다. 파이썬 예약어를 변수로 사용하면 다음과 같은 오류가 발생합니다.

```
>>> and = 1
SyntaxError: invalid syntax
```

파이썬 예약어를 모두 기억할 수 없으므로 프로그램을 실행할 때 이와 같은 오류가 발생하면 변수명을 제대로 사용했는지 확인하고 다른 변수명으로 수정한 후 다시 실행해 봅니다.

만약 파이썬 예약어를 확인하고 싶다면 파이썬 프롬프트에서 다음과 같은 방법을 활용합니다.

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif',
'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal',
'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

파이썬은 자체적으로 내장되어 있는 함수가 많이 있습니다. 예를 들어, `abs()`는 주어진 수를 절대값으로 변환하여 리턴하는 함수입니다.

만약 변수명으로 `abs`를 사용하게 되면 절대값을 리턴하는 함수인 `abs()`는 더 이상 사용할 수 없습니다.

```
>>> abs = 1
>>> abs(-3)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    abs(-3)
TypeError: 'int' object is not callable
```

이 오류는 변수명으로 사용된 `abs`를 함수처럼 호출하여 발생되는 오류입니다. 따라서 변수명으로 파이썬 내장 함수이름과 동일한 이름은 피해야 합니다.

입문

004

변수에 값 대입하기

- 학습 내용 : 다양한 값을 변수에 대입하는 방법을 배웁니다.
- 힌트 내용 : 값의 종류에 따라 파이썬은 알아서 변수 타입을 정해줍니다.

```
1: number1 = 1  
2: pi = 3.14  
3: flag = True  
4: char = 'x'  
5: chars = 'I love Python'
```

파이썬은 C/C++와는 달리 변수를 선언할 때 숫자형 자료인지 문자형 자료인지 자료형을 명시하지 않아도 됩니다. 파이썬 자료형에 대해서는 예제 006에서 따로 다루도록 합니다. 파이썬에서는 변수에 값을 대입하면 그 값의 자료형에 따라 변수의 자료형이 자동적으로 정해집니다.

- 1~2 ◆ 변수 `number1`에 정수 1을 대입했으므로 `number1`은 정수 자료가 되며 변수 `pi`에는 3.14를 대입해서 실수 자료가 됩니다.
- 3 ◆ `flag`에 `True`를 대입하여 참 또는 거짓을 나타내는 불린(boolean) 자료로 활용하고 있습니다.
- 4~5 ◆ 한 개의 문자를 변수에 대입하거나 여러 개의 문자로 구성된 문자열을 변수에 대입하여 한 문자 또는 문자열을 정의합니다.

변수에 정수를 대입하였다고 해서 이 변수가 정수형 자료로 고정되는 것은 아니며 변수에 값이 대입되는 시점에 변수의 자료형은 자동적으로 다시 정해집니다.

```
>>> number = 1          # number에 숫자 1을 대입. number는 정수형 자료임  
>>> number+2          # number+2를 계산  
3  
>>> number = 'one'      # number에 문자열 'one'을 대입. number는 문자열 자료임  
>>> number  
one
```

주석 처리하기(#)

입문
005

- 학습 내용 : 코드가 아닌 부분을 표시하는 주석 처리하는 방법에 대해 배웁니다.
- 힌트 내용 : 한 라인에서 '#' 기호 뒷부분은 모두 주석으로 간주합니다.

```

1: # 주석 처리 예시임
2: # 만든 날짜: 2016. 5. 30
3: a = 1           # a에 1을 대입함
4: b = 5           # b에 5를 대입함
5: print(a+b)     # a+b의 결과를 출력함
  
```

파이썬 코드 한 라인을 주석 처리하기 위해서는 문자 '#'으로 시작하면 됩니다. '#' 이전의 문자들은 주석 처리가 되지 않습니다.

코드를 작성할 때 주석을 잘 작성해 두면 차후에 코드를 다시 보거나 타인이 코드를 검토할 때 매우 중요한 정보로 활용됩니다. 그러므로 주석은 항상 달아주는 습관을 가지고 있는 것이 좋습니다.

특정 영역을 주석 처리하고자 하는 경우에는 삼중 따옴표를 이용해 주석으로 처리할 부분을 둘러싸면 됩니다.

```

# 삼중 따옴표로 특정 영역 주석 처리 예시
"""a = 1
b = 5
print(a+b)"""
a = 2
b = 6
print(a+b)
  
```

위 코드에서 삼중 따옴표로 둘러싸인 부분은 주석으로 처리되며, `a = 2` 이후부터 유효한 코드입니다.

입문

006



자료형 개념 배우기

- 학습 내용 : 데이터 타입이라고도 불리는 자료형에 대한 개념을 배웁니다.
- 힌트 내용 : 파이썬에서 자주 다루게 되는 자료형은 다섯 가지가 있습니다.

```
1: int_data = 1           # 정수 선언
2: float_data = 3.14      # 실수 선언
3: complex_data = 1+5j    # 복소수 선언
4: str_data1 = 'I love Python'  # 문자열 선언(영문)
5: str_data2 = "반갑습니다."  # 문자열 선언(한글)
6: list_data = [1, 2, 3]     # 리스트 선언
7: tuple_data = (1, 2, 3)    # 튜플 선언
8: dict_data = {0:'False', 1:'True'}  # 사전 선언
```

자료형 또는 데이터 타입이란 숫자, 문자 등과 같이 여러 종류의 데이터를 구분하기 위한 분류입니다.

파이썬은 다음과 같이 다섯 가지 기본 자료형이 있습니다.

1. 수치형 자료

수치형 자료는 수학에서 사용하는 수를 표현하는 자료형입니다. 수치형 자료는 다음과 같이 세 가지 상수를 취급하고 다릅니다.

- 정수형 상수: -1, 0, 1 과 같은 정수
- 실수형 상수: -0.7, 2.1 등과 같이 분수로 표현할 수 있는 유리수와 π (원주율), $\sqrt{2}$ 와 같은 무리수를 포함하는 실수
- 복소수형 상수: 실수부 + 허수부로 되어 있는 복소수

1-3 ◆ 각 변수에 정수형 상수 1, 실수형 상수 3.14, 복소수형 상수 $1+5j$ 를 대입한 예입니다. 파이썬에서 허수부는 j 로 표현하는 것에 유의합니다. 수치형 자료에 대한 내용은 예제 016 ~ 예제 018에서 다릅니다.

2. 문자열 자료

's', '안녕하세요', 'I love Python', '010-1234-5678' 등과 같이 한 글자 이상의 문자나 숫자, 기호로 구성된 자료입니다. 여기서 문자는 알파벳이나 한글과 같이 언어를 표현하는 글자를 의미하는 것으로 하겠습니다.

파이썬에서 문자열 자료의 선언 방법은 두 가지가 있습니다. 4라인처럼 ''로 둘러싸인 문자열을 변수에 대입하는 방법과, 5라인과 같이 " "로 둘러싸인 문자열을 변수에 대입하는 방법입니다. 문자열에 대한 내용은 예제 033에서 다룹니다.

◆ 4-5

3. 리스트 자료

리스트 자료는 '[']' 안에 임의의 객체를 순서 있게 나열한 자료형입니다. 각 요소는 콤마(,)로 구분합니다.

`list_data`의 요소는 1, 2, 3 숫자로만 되어 있지만 리스트의 각 요소는 임의의 자료형이나 객체가 될 수 있습니다. 리스트에 대한 내용은 예제 036에서 다릅니다.

◆ 6

4. 튜플 자료

튜플은 리스트와 비슷하지만 요소 값을 변경할 수 없다는 것이 리스트와 다릅니다. 튜플에 대한 내용은 예제 037에서 다릅니다.

◆ 7

5. 사전 자료

사전 자료는 '{ }' 안에 '키:값'으로 된 쌍이 요소로 구성된 순서가 없는 자료형입니다. 각 요소는 콤마로 구분하여 나열합니다. 사전 자료는 순서가 없으므로 인덱스로 값을 접근할 수 없고 키를 이용해 대응되는 값을 접근합니다. 사전에 대한 내용은 예제 038에서 다릅니다.

◆ 8

파이썬에서는 반복 가능한(iterable) 자료형이라는 개념이 있는데, 이는 예제 012에서 다룰 `for`문에서 `in` 다음에 사용될 수 있는 자료형을 말합니다. 파이썬에서 반복 가능한 자료형으로 취급되는 것은 문자열, 리스트, 튜플, 사전 이외에 다양한 객체 형태로 존재합니다. 반복 가능한 자료형은 파이썬 내장함수 `list()`를 이용해 리스트 자료로 변환할 수 있습니다. 반복 가능한 자료형에 대해서는 앞으로 계속 언급될 것이므로 여기서는 그 개념만 이해하도록 합니다.

입문

007

자료형 출력 개념 배우기(**print**)

- 학습 내용 : 다양한 자료를 화면에 출력하는 방법에 대해 배웁니다.
- 힌트 내용 : `print()`는 다양한 자료형을 다양한 방법으로 화면에 출력할 수 있는 함수입니다.

소스 : 007.py

```
1: a = 200
2: msg = 'I love Python'
3: list_data = ['a', 'b', 'c']
4: dict_data = {'a':97, 'b':98}
5: print(a)
6: print(msg)
7: print(list_data)
8: print(list_data[0])
9: print(dict_data)
10: print(dict_data['a'])
```

`print()`는 인자로 입력된 자료형 및 객체 값을 화면에 출력합니다. `print()`의 인자는 임의의 객체가 가능합니다.

- 1~4 ◆ 변수에 수치형 자료, 문자열, 리스트, 사전 자료를 정의해서 대입합니다.
- 5 ◆ 변수 `a`의 값을 화면에 출력합니다. 변수 `a`에는 숫자 200이 대입되어 있으므로 200이 출력됩니다.
- 6 ◆ 변수 `msg`의 값을 화면에 출력합니다. 변수 `msg`에는 'I love Python'이라는 문자열이 대입되어 있으므로 'I love Python'을 출력합니다.
- 7~8 ◆ 리스트 자료인 `list_data`와 `list_data`의 첫 번째 요소를 출력합니다.
- 9~10 ◆ 사전 자료인 `dict_data`와 `dict_data`에서 키가 `a`인 값을 출력합니다.

IDLE 텍스트 에디터에 이 코드를 작성하고 [F5]를 눌러 실행하면 다음과 같은 결과가 화면에 출력됩니다.



```
200
I love Python
['a', 'b', 'c']
a
{'b': 98, 'a': 97}
97
```

`print()`는 기본적으로 인자로 입력된 값을 화면에 출력한 후 줄바꿈을 합니다. 다시 말해, `print()`는 값을 출력하고 다음에 출력할 때는 줄을 바꾸어 출력합니다. `print()`는 기본적으로 출력되는 값 뒤에 항상 '`\n`'을 추가한다는 의미입니다. '`\n`'은 줄바꿈을 의미하는 문자입니다. 만약 줄바꿈 문자 '`\n`'을 추가하지 않으려면 다음의 예와 같이 `print()`의 두 번째 인자로 `end = "`를 지정하면 됩니다.

```
>>> print('# ', end='')
>>> print('#')
# #
```

입문

008

들여쓰기 개념 배우기

- 학습 내용 : 파이썬에서 들여쓰기란 어떤 의미인지 배웁니다.
- 힌트 내용 : 파이썬에는 실행 코드 부분을 묶어주는 {} 괄호가 없습니다.

소스 : 008.py

```
1: listdata = ['a', 'b', 'c']
2: if 'a' in listdata:
3:     print('a가 listdata에 있습니다.')
4:     print(listdata)
5: else:
6:     print('a가 listdata에 존재하지 않습니다.)
```

파이썬은 다른 프로그래밍 언어와 달리 if, for, while 등과 같은 제어문이나 함수 및 클래스에서 실행 코드 부분을 구분해주는 팔호 {}가 없습니다. 대신 들여쓰기(indentation)로 팔호 {}를 대신합니다. **Space Bar**로 들여쓰기를 할 수 있고 **Tab**으로 들여쓰기를 할 수 있습니다.

파이썬에서 제어문이나 함수이름, 클래스 이름 뒤에 콜론(:)으로 제어문, 함수이름, 클래스 이름의 끝을 표시하며 ‘:’ 다음에 실행 코드를 작성합니다.

예를 들어, 실행 코드가 한 라인일 경우 다음과 같이 코드를 작성할 수 있습니다.

```
if 'a' in listdata: print('a가 listdata에 있습니다.)
```

위 예에서는 “if 'a' in listdata:”가 제어문이며 콜론(:)으로 제어문의 끝을 알리고, 이 제어문의 실행 코드는 print('a가 listdata에 있습니다.)입니다.

그런데 실행 코드는 한 라인 이상인 경우가 대부분이라 콜론(:) 다음에 [Enter]를 눌러 라인을 바꾼 후 실행 코드를 작성하게 됩니다. 이때 실행 코드는 반드시 들여쓰기를 해야 합니다.

```
if 'a' in listdata:  
    실행 코드
```

파이썬 들여쓰기는 다음과 같은 기본 규칙을 가집니다.

1. 가장 바깥쪽의 실행 코드는 들여쓰기 없이 시작해야 함

만약 예제 코드에서 다음과 같이 공백이 있는 상태로 if문을 작성하게 되면 “SyntaxError: unexpected indent” 오류가 발생합니다.

```
(공백)listdata = ['a', 'b', 'c']  
if 'a' in listdata:  
    ...
```

2. 콜론(:) 다음 라인부터 시작하는 실행 코드는 들여쓰기 간격이 모두 동일해야 함

다음과 같이 if문 실행 코드의 들여쓰기 간격이 동일하지 않으면 “SyntaxError: unexpected indent” 오류가 발생합니다.

```
listdata = ['a', 'b', 'c']  
if 'a' in listdata:  
    print('a가 listdata에 있습니다.')  
    print(listdata)          # 오류가 발생하는 지점  
else:  
    print('a가 listdata에 존재하지 않습니다.')
```

입문

009

if문 개념 배우기 ① (if~else)



- 학습 내용 : 어떤 조건을 참과 거짓으로 판단하기 위한 if문의 개념을 배웁니다.
- 힌트 내용 : 참과 거짓을 구분하여 코드를 실행하려면 if~else를 사용합니다.

소스 : 009.py

```
1: x = 1
2: y = 2
3: if x >= y:
4:     print('x가 y보다 크거나 같습니다.')
5: else:
6:     print('x가 y보다 작습니다.')
```

코드를 작성하다 보면 조건에 따라 수행하는 일을 달리해야 하는 경우가 있습니다. 조건이 참인지 거짓인지 검사를 하고, 참인 경우에는 이 일을 하고, 거짓인 경우에는 저 일을 하라는 식으로 처리해야 하는 경우입니다.

if문은 조건이 참인지 아닌지 판단하여 코드를 수행할 때 사용하는 조건문입니다. 파이썬 if문의 기본적인 사용은 다음과 같습니다.

if 조건:

 실행 코드 1

else:

 실행 코드 2

`if` 뒤의 조건이 참이면 실행 코드 1을 수행하고, 조건이 거짓이면 실행 코드 2를 수행합니다.

변수 `x, y`에 각각 1, 2를 대입합니다.

◆ 1~2

`x`가 `y`보다 크거나 같으면 '`x`가 `y`보다 크거나 같습니다.'를 출력합니다.

◆ 3~4

`x`가 `y`보다 작으면 '`x`가 `y`보다 작습니다.'를 출력합니다.

◆ 5~6

만약 조건이 참인 경우만 특정 코드를 실행하는 로직을 구현하려면 다음과 같이 `else`문 없이 `if`문만 사용해서 조건을 체크하면 됩니다.

```
if 조건:  
    실행 코드
```

입문

010



if문 개념 배우기 ② (if~elif)

- 학습 내용 : 다양한 조건을 판단하기 위한 if문을 배웁니다.
- 힌트 내용 : 여러 가지 조건들을 각각 검사하고 판단해야 할 경우 if~elif를 사용합니다.

소스 : 010.py

```
1: x = 1
2: y = 2
3: if x > y:
4:     print('x가 y보다 큽니다.')
5: elif x < y:
6:     print('x가 y보다 작습니다.')
7: else:
8:     print('x와 y가 같습니다.)
```

여러 개의 조건을 순차적으로 체크하고 해당 조건이 참이면 특정 로직을 수행하고자 할 때 if ~ elif 문을 사용합니다. if ~ elif문의 기본적인 사용은 다음과 같습니다.

```
if 조건 1:
    실행 코드 1
elif 조건 2:
    실행 코드 2
else:
    실행 코드 3
```

조건 1이 참이면 실행 코드 1을, 조건 2가 참이면 실행 코드 2를 수행합니다. 조건 1과 조건 2가 모두 거짓이면 실행 코드 3을 수행합니다. 체크하고자 하는 조건이 여러 개인 것을 제외하면 if~else문과 개념이 동일합니다.

- 3-8 ◆ 예제 코드는 x가 y보다 큰지 확인하고 조건이 거짓이면 x가 y보다 작은지 확인합니다. 이 결과도 거짓이면 x와 y는 값이 같으므로 'x와 y가 같습니다.'를 출력합니다.

for문 개념 배우기 ① (for)



- 학습 내용 : 주어진 범위에서 반복적으로 코드를 실행하는 for문의 개념을 배웁니다.
- 힌트 내용 : 반복 실행 코드는 for문을 이용하여 구현합니다.

소스 : 011.py

```

1: scope = [1, 2, 3, 4, 5]
2: for x in scope:
3:     print(x)

```

예제 코드를 실행하면 1에서 5까지 정수를 순차적으로 출력합니다.

for문은 특정 범위의 자료나 객체에 대해 처음부터 끝까지 하나씩 추출하여 특정 코드를 반복적으로 수행하기 위해 가장 많이 사용되는 반복문입니다. for문의 기본적인 구문은 다음과 같습니다.

for 변수 **in** 범위:
반복으로 실행할 코드

for문의 범위로 사용되는 것은 시퀀스 자료형 또는 반복 가능한 자료이어야 하며 다음과 같은 객체들이 있습니다.

- 문자열
- 리스트나 튜플
- 사전
- range()
- 그 외 반복 가능한 객체

문자열을 범위로 지정한 예

```
str = 'abcdef'  
for c in str:  
    print(c)
```

리스트를 범위로 지정한 예

```
list = [1, 2, 3, 4, 5]  
for c in list:  
    print(c)
```

사전을 범위로 지정한 예

```
ascii_codes = {'a':97, 'b':98, 'c':99}  
for c in ascii_codes:  
    print(c)
```

range() 함수를 범위로 지정한 예

```
for c in range(10):  
    print(c)
```

위의 예제를 실제로 돌려보면서 결과가 어떻게 나오는지 직접 확인해보기 바랍니다.

for문 개념 배우기 ② (for~continue~break)

입문

012

- 학습 내용 : for 반복문 내에서 continue와 break의 역할에 대해 이해합니다.
- 힌트 내용 : for 반복문 내에서 continue를 만나면 그 다음 반복 실행으로 넘어가며, break를 만나면 for 반복문을 완전히 벗어납니다.

 소스 : 012.py

```

1: scope = [1, 2, 3, 4, 5]
2: for x in scope:
3:     print(x)
4:     if x < 3:
5:         continue
6:     else:
7:         break

```

for문을 이용해 반복문을 수행하다가 어떤 조건에 따라 반복문을 계속 수행하고, 아니면 반복문을 벗어나야 하는 경우가 있습니다. 이 경우 for 반복문 안에서 continue를 만나면 이후 코드는 수행하지 않고 다음 반복문을 수행하게 되며, break를 만나면 for 반복문을 탈출하게 됩니다.

for 변수 in 범위:

```

...
continue    # 다음 반복문 수행
...
break      # for 반복문 탈출

```

예제 코드는 1~5까지 정수에 대해 코드를 반복하는데, 화면에 해당 정수를 출력한 후 그 수가 3보다 작으면 그 다음 숫자를 출력하고 3보다 크거나 같으면 for 반복문을 탈출하여 종료합니다.

따라서 예제 코드를 실행하면 1, 2, 3까지 숫자가 출력됩니다. 이 예제 코드는 다음과 같이 continue 없이 break만으로 동일한 로직을 작성할 수 있습니다.

```
scope = [1, 2, 3, 4, 5]
for x in scope:
    print(x)
    if x >= 3:
        break
```

for문 개념 배우기 ③ (for~else)

입문

013

- 학습 내용 : for 반복문을 완전히 수행했을 때만 실행하는 부분을 정의하는 방법을 배웁니다.
- 힌트 내용 : for~else에서 else 뒤의 실행 코드는 for 반복문을 모두 성공적으로 수행해야만 실행됩니다.



소스 : 013.py

```

1: scope = [1, 2, 3]
2: for x in scope:
3:     print(x)
4:     break
5: else:
6:     print('Perfect')

```

for 반복문이 break에 의해 중단됨이 없이 정상적으로 모두 실행되어야만 특정 코드를 실행하게 할 경우 for ~ else를 사용합니다.

for 변수 **in** 범위:

반복 실행 코드

else:

for 구문이 모두 실행되었을 때 실행할 코드

else:로의 진입은 for 반복문에서 break 등에 의해 중간에 중단됨이 없이 정상적으로 실행이 다 되었을 경우입니다. 예제 코드의 실행 결과는 다음과 같습니다.

```

1
2
3
Perfect

```

소스 013.py에서 break에 주석 처리하고 코드를 실행하면 화면에 1만 출력됩니다.

입문

014

while문 개념 배우기 (while~continue~break)

- 학습 내용 : 특정 조건이 만족하는 경우 반복 수행하게 하는 while문의 개념을 배웁니다.
- 힌트 내용 : for는 특정 범위에서, while은 특정 조건에서 코드를 반복 수행하게 합니다.

소스 : 014.py

```
1: x = 0
2: while x < 10:
3:     x = x+1
4:     if x < 3:
5:         continue
6:     print(x)
7:     if x > 7:
8:         break
```

for문이 범위가 지정된 자료나 반복 가능한 객체를 이용해 반복문을 수행하는 것이라면 while문은 특정 조건을 만족하는 경우 지속적으로 반복을 수행하는 반복문입니다.

while문의 기본적인 구문은 다음과 같습니다.

while 조건:

반복 실행 코드

continue # while 구문 처음으로 이동하여 반복문 계속

...

break # while 구문을 탈출함

while 다음에 놓인 조건이 거짓일 때까지 while 반복문을 계속 실행합니다. while 반복문 내에서 continue 키워드를 만나면 while 반복문 처음으로 돌아와 반복문을 계속 수행합니다. 하지만 while 반복문 내에서 break 키워드를 만나면 while 반복문을 탈출하게 됩니다.

예제 코드는 **x**의 초기값을 0으로 설정하고, 변수 **x**의 값이 10보다 작으면 while 반복문을 계속 수행하는 코드입니다. **x**의 초기값이 0이므로 while의 조건이 참이 되며 while 구문으로 진입하게 됩니다.

◆ 1-2

x의 값을 1 증가시키고 **x**가 3보다 작으면 continue를 통해 while 반복문 처음으로 돌아가게 합니다. 만약 **x**가 3보다 크거나 같으면 **x**값을 출력하고 **x**가 7보다 큰 값인지 체크한 후, 7보다 크면 while 반복문을 탈출합니다.

◆ 3-8

예제 코드를 실행하면 3, 4, 5, 6, 7, 8을 출력합니다.

while문은 특정 조건을 만족할 때까지 반복 실행하는 무한 루프를 구현하는 경우에 많이 사용됩니다. 예를 들면, 사용자가 키보드로 [Esc]를 누를 때까지 프로그램을 계속 실행하게 하는 경우입니다. 다음의 코드는 $1 + 2 + 3 + \dots + n$ 과 같이 정수 1부터 n까지 더할 때 그 합이 10만보다 커지게 되는 n을 구하는 것입니다.

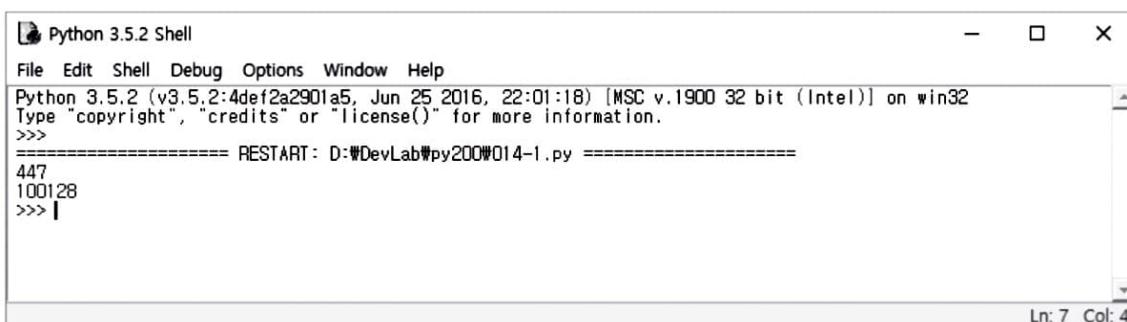
소스 : 014-1.py

```
x = 1
total = 0
while 1:
    total = total + x
    if total > 100000:
        print(x)
        print(total)
        break
    x = x + 1
```

보통 프로그래밍 언어에서 참은 1로, 거짓은 0으로 정의합니다. 파이썬도 참은 1, 거짓은 0으로 정의하고 있습니다. 좀 더 염밀하게 말하면 0이 아닌 값은 모두 참, 0은 거짓으로 정의합니다. 참과 거짓은 True와 False를 사용하는 것을 권장하며 이에 대해서는 예제 022에서 다루도록 합니다.

while 1:은 조건 자체가 참이므로 while 반복문을 무한 반복하게 됩니다. 이런 경우 while 반복문을 탈출하기 위해서는 break를 활용하는 방법 밖에 없습니다. 무한 루프를 돌면서 변수 total 값에 x 값을 더하고 total 값을 갱신합니다. total 값이 10만보다 커지면 x 값을 total 값을 출력하고 while 반복문을 탈출합니다. total 값이 10만보다 작으면 x를 1 증가시키고 while 반복문을 계속 실행합니다.

이 코드를 실행하면 다음과 같이 x는 447, total은 100128이 결과로 나옵니다.



The screenshot shows a window titled "Python 3.5.2 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main window displays the Python interpreter's prompt (>>>) followed by the output of a script named "014-1.py". The script's output is as follows:

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
=====
RESTART: D:\DevLab\py200\014-1.py =====
447
100128
>>> |
```

In the bottom right corner of the window, it says "Ln: 7 Col: 4".

None 개념 배우기

입문

015



- 학습 내용 : 파이썬의 특별한 상수 None에 대한 개념을 배웁니다.
- 힌트 내용 : None은 아무것도 없다는 의미의 상수입니다.

소스 : 015.py

```

1: val = None
2: condition = 1
3: if condition == 1:
4:     val = [1, 2, 3]
5: else:
6:     val = 'I love Python'

```

파이썬은 None이라는 특별한 상수가 있습니다. 파이썬 문서에서 설명하는 None은 다음과 같습니다.

None은 Types.NoneType의 유일한 값으로, 값이 존재하지 않는 변수에 대입하여 이 변수에 아무런 값이 없다는 것을 나타내기 위해 주로 활용된다.

다시 말해 None이 대입된 변수는 아무런 값도 없는 빈 깡통 변수라고 생각하면 됩니다. Val = None으로 선언한 경우, val은 아무런 값도 가지지 않는 변수이므로 이 변수로는 할 수 있는 것이 아무것도 없습니다.

하지만 None으로 지정된 변수에 값이 있는 임의의 자료형을 대입하여 활용할 수 있습니다.

어떤 변수에 대입할 자료형을 결정하지 않은 상태로 선언할 때 None을 대입하거나, 함수에서 아무 값도 리턴하지 않고 끝낼 때 사용하기도 합니다. 함수 로직에서 예외가 발생하거나 오류가 발생하여 None을 리턴하여 종료하면 이 함수를 호출한 쪽에서 호출한 함수가 실행 도중 오류가 발생하여 비정상적으로 종료되었음을 알 수 있게 합니다.

예제는 condition의 값에 따라 val에 리스트 또는 문자열을 대입하는 예입니다. condition의 값이 1이면 val에 리스트 [1, 2, 3]을 대입하고 condition의 값이 1이 아니면 변수 val에 문자열 'I love Python'을 대입합니다.

2

P A R T

초급

Python 프로그래밍 기초 다지기

초급

016

정수형 자료 이해하기



- 학습 내용 : 정수형 자료를 이해하고 선언하는 방법에 대해 배웁니다.
- 힌트 내용 : -1, 0, 1, 2와 같은 수를 정수라고 하며, 목적에 따라 10진수, 2진수, 16진수 등으로 변환하여 활용할 수 있습니다.

```
1: int_data = 10
2: bin_data = 0b10
3: oct_data = 0o10
4: hex_data = 0x10
5: long_data = 1234567890123456789
6: print(int_data)
7: print(bin_data)
8: print(oct_data)
9: print(hex_data)
10: print(long_data)
```

정수형 자료는 정수형 상수를 취합니다. 프로그래밍에서 일반적으로 사용되는 정수형 상수는 10진수입니다. 그런데 컴퓨터는 0과 1로 표현되는 2진수로 처리하기 때문에 코드에서 8진수나 16진수를 사용하는 것이 편리한 경우가 있습니다.

- 2-5 ◆ 파이썬 3에서 2진수는 0b, 8진수는 0o, 16진수는 0x로 시작하는 수입니다. 일반적으로 프로그래밍 언어에서 지원하는 정수형 상수의 범위는 $-2147483647 \sim 2147483647$ 까지이지만 파이썬의 정수형 상수의 최소값, 최대값은 존재하지 않고, 메모리가 허용하는 범위에서 지원 가능한 수를 사용할 수 있습니다.
- 6-10 ◆ 정수형 상수를 대입한 변수를 출력하면 다음과 같이 모두 10진수로 나옵니다.



10
2
8
16
1234567890123456789



N O T E | 1바이트와 16진수 |

일반적으로 컴퓨터에 기록되는 데이터의 최소 단위는 8비트 또는 1바이트입니다. 1비트는 한 자리 2진수이므로 8비트는 8자리 2진수로 표현됩니다. 8진수는 3자리 2진수를 한 자리수로 표현 가능하며, 16진수는 4자리 2진수를 한 자리수로 표현이 가능합니다. 따라서 1바이트 데이터는 두 자리 16진수로 표현이 가능하므로 컴퓨터에 기록된 대부분의 데이터는 16진수로 표현이 가능합니다.

초급

017

실수형 자료 이해하기



- 학습 내용 : 실수형 자료를 이해하고 선언하는 방법에 대해 배웁니다.
- 힌트 내용 : 파이썬에서 소수점으로 나타내는 상수는 실수형 상수로 간주합니다.

소스 : 017.py

```
1: f1 = 1.0
2: f2 = 3.14
3: f3 = 1.56e3
4: f4 = -0.7e-4
5: print(f1)
6: print(f2)
7: print(f3)
8: print(f4)
```

실수는 소수로 나타낼 수 있는 유리수와 원주율이나 $\sqrt{2}$ 와 같은 소수로 표현할 수 없는 무리수로 구성된 수 집합입니다.

- 1-2 ◆ 변수에 소수로 표현한 값을 대입하면 이 변수는 실수형 자료로 취급됩니다. f1은 값이 1이지만 실수형 자료로 취급됩니다. f2도 f1과 마찬가지로 3.14라는 값을 가지는 실수형 자료가 됩니다.
- 3-4 ◆ 실수형 상수를 표현하는 또 다른 방법은 e를 사용하는 것입니다. e는 10의 거듭제곱을 나타냅니다. e2는 10^2 을 나타내며, e-3은 10^{-3} 을 나타냅니다. 예제 코드의 1.56e3은 1.56×10^3 을 나타냅니다. 따라서 f3의 값은 1560.0이 됩니다. f4는 -0.7×10^{-4} 입니다. 파이썬은 이 값을 -7×10^{-5} 로 변경하여 다루게 됩니다.

예제 코드의 실행 결과는 다음과 같습니다.



결과

```
1.0
3.14
1560.0
-7e-05
```



N O T E | 실수형 자료와 정수형 자료의 연산 |

실수형 자료와 정수형 자료를 연산해서 나오는 결과는 모두 실수형 자료가 됩니다. IDLE을 구동해서 다음과 같은 코드로 간단하게 확인해 봅니다.

```
>>> i = 2; f = 4.0
>>> print(f+i)
6.0
>>> print(f*i)
8.0
>>> print(f/i)
2.0
>>> print(4/2)
2.0
```

`i`는 정수 2, `f`는 실수 4.0을 대입한 변수입니다. 사실 연산의 결과가 모두 소수점으로 표현되어 있으므로 그 결과가 실수형 자료임을 알 수 있습니다.

참고로 나눗셈의 결과는 그 자료형에 관계없이 실수형 자료가 됩니다.

```
>>> print(4/2)
2.0
```

정수 4를 정수 2로 나눈 결과는 실수형 상수 2.0이 됩니다.

초급

018

복소수형 자료 이해하기



- 학습 내용 : 복소수형 상수를 이해하고 선언하는 방법에 대해 배웁니다.
- 힌트 내용 : 복소수는 실수부와 허수부로 되어 있고, 허수부는 숫자 뒤에 문자 j를 붙입니다.

소스 : 018.py

```
1: c1 = 1+7j  
2: print(c1.real); print(c1.imag)  
3: c2 = complex(2, -3)  
4: print(c2)
```

복소수형 상수는 실수부 + 허수부로 되어 있는 수입니다. 우리가 배운 수학에서의 허수부는 i를 이용해 표현하고 있지만 파이썬에서는 j를 이용합니다. 실수부와 허수부를 구성하는 수는 실수형 상수입니다.

- ◆ 변수 c1에 복소수형 상수 $1 + 7j$ 를 대입합니다.
- ◆ 복소수형 자료에서 실수부만 취하려면 real을 이용합니다. 복소수형 자료에서 허수부만 취하려면 imag를 이용합니다. 실수부 1과 허수부 $7j$ 를 구성하는 수 1과 7은 모두 실수형 상수로 취급하므로 c1.real과 c1.imag는 각각 1.0, 7.0이 됩니다.
- ◆ complex()를 이용해 복소수형 상수를 구성할 수 있습니다. complex(2, 3)은 실수부가 2, 허수부가 3인 복소수를 만듭니다. 이 값을 변수 c2에 대입하므로 c2의 값은 $2 + 3j$ 가 됩니다.

예제 코드를 실행하면 다음과 같은 결과가 출력됩니다.



1.0
7.0
 $(2+3j)$

대입 연산자 이해하기(=)

초급

019

- 학습 내용 : 변수에 값을 대입하는 대입 연산자에 대해 이해합니다.
- 힌트 내용 : 대입 연산자는 등호 '='를 이용합니다.



소스 : 019.py

```

1: a = 1
2: b = 2
3: ret = a+b
4: print('a와 b를 더한 값은 ', end="")
5: print(ret, end="")
6: print('입니다.')

```

변수에 값을 대입하는데 사용되는 기호는 =(등호)입니다. 수학에서 =는 = 왼쪽의 값과 오른쪽의 값이 같다는 의미를 가지지만, 파이썬을 포함한 컴퓨터 프로그래밍 언어에서 =는 = 왼쪽의 변수에 = 오른쪽의 값을 대입한다는 의미입니다.

파이썬에서 두 값이 같다는 것을 나타내는 등호 연산자는 = 두 개가 연속으로 되어 있는 ==입니다. 연산자 ==에 대한 내용은 예제 023에서 다룹니다.

변수 a와 b에 각각 정수형 상수 1과 2를 대입합니다.

◆ 1~2

변수 ret에 a와 b를 더한 값을 대입합니다. 변수 a와 b의 값이 각각 1과 2이므로 ret은 3이 됩니다.

◆ 3

'a와 b를 더한 값은 '을 줄바꿈 문자 \n을 생략하고 출력하고, ret의 값을 출력한 후, '입니다.'를 출력합니다.

◆ 4~6

예제 코드를 실행한 결과는 다음과 같습니다.

결과

a와 b를 더한 값은 3입니다.

초급

020

사칙 연산자 이해하기 (+, -, *, /, **)

- 학습 내용 : 파이썬에서 더하기, 빼기, 곱하기, 나누기와 같은 사칙연산에 대해 이해합니다.
- 힌트 내용 : 더하기, 빼기, 곱하기, 나누기는 각각 +, -, *, / 기호를 이용하고 거듭제곱은 **를 이용합니다.

 소스 : 020.py

```

1: a = 2
2: b = 4
3: ret1 = a+b           # ret1 = 2+4 = 6
4: ret2 = a-b           # ret2 = 2-4 = -2
5: ret3 = a*b           # ret3 = 2*4 = 8
6: ret4 = a/b           # ret4 = 2/4 = 0.5
7: ret5 = a**b          # ret5 = 2**4 = 16
8: ret6 = a+a*b/a      # ret6 = 2+2*4/2 = 6
9: ret7 = (a+b)*(a-b)   # ret7 = (2+4)*(2-4) = -12
10: ret8 = a*b**a       # ret8 = 2*4**2 = 32

```

수학에서 사칙 연산은 덧셈, 뺄셈, 곱셈, 나눗셈을 말합니다. 파이썬에서 말하는 사칙 연산도 이와 동일합니다. 단 곱셈과 나눗셈을 나타내는 기호는 각각 *와 /입니다. **는 거듭제곱을 나타내는 연산 기호입니다. $2^{**}4$ 는 2^4 을 의미하며 값은 16이 됩니다.

- 1-2 ◆ 변수 a와 b에 각각 정수형 상수 2와 4를 대입합니다.
- 3 ◆ 정수형 자료 a와 b를 더한 값을 변수 ret1에 대입합니다. ret1의 값은 6입니다.
- 4 ◆ 정수형 자료 a에서 b를 뺀 값을 변수 ret2에 대입합니다. ret2의 값은 -2입니다.
- 5 ◆ 정수형 자료 a와 b를 곱한 값을 변수 ret3에 대입합니다. ret3의 값은 8입니다.
- 6 ◆ 정수형 자료 a에서 b를 나눈 값을 변수 ret4에 대입합니다. 파이썬 3은 자료형과 상관없이 나눗셈의 경우 실수형으로 결과를 리턴합니다. ret4의 값은 0.5입니다.
- 7 ◆ 정수형 자료 a를 b번 거듭제곱하여 변수 ret5에 대입합니다. ret5의 값은 16입니다.

◆ 8
사칙 연산자가 섞여 있을 때 연산하는 우선 순위는 곱셈과 나눗셈을 먼저 수행하고 덧셈, 뺄셈을 수행합니다. 곱셈과 나눗셈은 연산 우선 순위가 같으므로 순서대로 연산을 하고, 덧셈과 뺄셈도 서로 연산 우선 순위가 같으므로 순서대로 연산을 수행하면 됩니다. 따라서 $a+a \cdot b/a$ 는 $a \cdot b/a$ 를 먼저 계산하고 이 값을 a 와 더하는 순서가 됩니다.

괄호()로 묶여 있는 부분을 먼저 계산한 후 사칙 연산을 수행합니다. $(a+b) \cdot (a-b)$ 는 $(a+b)$ 와 $(a-b)$ 를 먼저 계산하고 두 값을 곱합니다. ◆ 9

거듭제곱은 사칙 연산보다 연산 순위가 더 우선합니다. $a \cdot b^{**}a$ 는 $b^{**}a$ 를 먼저 계산하고 a 와 곱하여 값을 구합니다. ◆ 10

초급

021

연산자 축약 이해하기 ($+=$, $-=$, $*=$, $/=$)



• 학습 내용 : 변수에 값을 사칙 연산하여 그 결과를 동일한 변수에 대입할 때 사용되는 연산자 축약에 대해 이해합니다.

• 힌트 내용 : $a = a+2$ 를 연산자 축약으로 표시하면 $a += 2$ 가 됩니다.

- | | |
|------------------|--|
| 1: $\alpha = 0$ | |
| 2: $\alpha += 1$ | # $\alpha = \alpha + 1$, α 의 값은 1 |
| 3: $\alpha -= 5$ | # $\alpha = \alpha - 5$, α 의 값은 -4 |
| 4: $\alpha *= 2$ | # $\alpha = \alpha * 2$, α 의 값은 -8 |
| 5: $\alpha /= 4$ | # $\alpha = \alpha / 4$, α 의 값은 -2.0 |

변수 α 에 1을 더하고 그 결과를 다시 변수 α 에 대입할 때 다음과 같은 식이 됩니다.

$$\alpha = \alpha + 1$$

이와 같이 어떤 변수와 어떤 값을 사칙 연산한 결과를 다시 동일한 변수에 대입할 때 다음과 같이 연산자 축약으로 표현할 수 있습니다.

$$\alpha += 1$$

- 1~4 ◆ α 에 정수형 상수 0을 대입하고, α 에 1을 더한 결과를 다시 α 에 대입합니다. α 에 5를 뺀 결과를 다시 α 에 대입하고, α 에 2를 곱한 값을 α 에 대입합니다. 여기까지 계산하면 α 의 값은 정수형 값 -8이 됩니다.
- 5 ◆ α 를 정수형 상수 4로 나누고 그 결과를 α 에 대입합니다. 나눗셈을 수행하였으므로 α 는 실수형 상수를 담은 변수가 됩니다. α 의 최종값은 -2.0이 됩니다.

True와 False 이해하기

초급

022

- 학습 내용 : 참을 나타내는 True와 거짓을 나타내는 False에 대해 이해합니다.
- 힌트 내용 : 조건을 판단해서 그 조건이 참이면 True, 거짓이면 False를 리턴합니다.



소스 : 022.py

```

1: a = True
2: b = False
3: print(a == 1)           # True가 출력됨
4: print(b != 0)           # False가 출력됨

```

파이썬에서 참과 거짓을 나타내는 상수는 True와 False입니다. True는 1, False는 0의 값을 가집니다.

참과 거짓을 나타낼 때 True와 False로 표현하면 더 직관적이고 프로그램 코드의 가독성을 높일 수 있습니다.

a에 True를, b에 False를 대입합니다.

◆ 1-2

True 값이 대입된 a가 1과 같은 값인지 체크합니다. 같은 값이면 True를, 다른 값이면 False를 출력하게 되는데, 결과는 True입니다.

◆ 3

False 값이 대입된 b가 0과 다른 값인지 체크합니다. 다른 값이면 True를, 같은 값이면 False를 출력하게 되는데, 결과는 False입니다.

◆ 4

코드에 무한 루프 로직을 구현할 때 다음과 같이 while 반복문을 만듭니다.

while True:

(무한 반복 실행 코드)

...

if 조건 == True:
 break

조건이 만족하면 무한루프를 탈출함

초급

023

관계 연산자 이해하기 $(==, !=, <, \leq, >, \geq)$

- 학습 내용 : 주어진 두 개의 값을 비교하는 관계 연산자에 대해 이해합니다.
- 힌트 내용 : 두 개의 값을 비교하면 같거나 다르거나 또는 크거나 작거나입니다.

소스 : 023.py

```
1: x = 1; y = 2
2: str1 = 'abc'; str2 = 'python'
3: print(x == y)           # False가 출력됨
4: print(x != y)          # True가 출력됨
5: print(str1 == str2)     # False가 출력됨
6: print(str2 == 'python') # True가 출력됨
7: print(str1 < str2)      # True가 출력됨
```

관계 연산자는 두 개의 값을 서로 비교하여 참인지 거짓인지 판단할 때 사용되는 연산자입니다. 파이썬에서 사용 가능한 관계 연산자는 다음과 같습니다.

관계 연산자	의미
$A == B$	A와 B가 같으면 참
$A != B$	A와 B가 다르면 참
$A < B$	A가 B보다 작으면 참
$A \leq B$	A가 B보다 작거나 같으면 참
$A > B$	A가 B보다 크면 참
$A \geq B$	A가 B보다 크거나 같으면 참

- ◆ 예제에서 x, y에 각각 정수형 상수 1과 2를 대입하고, str1과 str2에는 문자열 'abc'와 'python'을 각각 대입합니다.
- ◆ x와 y가 같은 값인지 검사하고 결과를 출력합니다. 결과는 False입니다.
- ◆ x와 y가 다른 값인지 검사하고 결과를 출력합니다. 결과는 True입니다.

파이썬 관계 연산자는 숫자뿐만 아니라 모든 자료형에 적용할 수 있습니다. 5라인 이후는 관계 연산자를 이용해 문자열을 비교하는 예를 보인 것입니다.

문자열이 대입된 str1과 str2가 서로 같은지 비교하고 결과를 출력합니다. 결과는 False입니다. ◆ 5

str2와 'python'이 서로 같은지 비교하고 결과를 출력합니다. 결과는 True입니다. ◆ 6

str1이 str2보다 작은지 비교하고 결과를 출력합니다. 문자열의 크기 비교는 문자열의 사전 순서로 비교합니다. 'abc'가 'python'보다 사전 순서가 앞이므로 결과는 True입니다. ◆ 7

다른 관계 연산자들도 파이썬 모든 자료형에 적용하여 비교할 수 있습니다.

초급

024

논리 연산자 이해하기 (and, or, not)

- 학습 내용 : 참 또는 거짓인 두 개의 값을 비교하는 논리 연산자의 개념을 이해합니다.
- 힌트 내용 : “새는 날개를 가진 동물이고 알을 낳는 동물이다.”는 참입니다.

소스 : 024.py

```
1: bool1 = True; bool2 = False; bool3 = True; bool4 = False
2: print(bool1 and bool2)          # False가 출력됨
3: print(bool1 and bool3)          # True가 출력됨
4: print(bool2 or bool3)           # True가 출력됨
5: print(bool2 or bool4)           # False가 출력됨
6: print(not bool1)                # False가 출력됨
7: print(not bool2)                # True가 출력됨
```

논리 연산자는 참, 거짓으로 되어 있는 두 개의 값을 비교하여 참 또는 거짓으로 결과를 내놓는 연산자입니다. 파이썬에서 사용 가능한 논리 연산자는 다음과 같습니다.

논리 연산자	의미
A and B	A와 B가 모두 참이면 참
A or B	A, B 중 하나 이상이 참이면 참
not A	A 논리값의 반대

- ◆ bool1과 bool3은 True, bool2와 bool4는 False를 각각 대입합니다.
- ◆ bool1과 bool2의 and 연산 결과를 출력합니다. bool1이 True이고 bool2가 False이므로 and 연산 결과는 False입니다.
- ◆ bool1과 bool3의 and 연산 결과를 출력합니다. bool1과 bool3 모두 True이므로 and 연산 결과는 True입니다.
- ◆ bool2와 bool3의 or 연산 결과를 출력합니다. bool2가 False이고 bool3이 True이므로 or 연산 결과는 True입니다.

bool2와 bool4의 or 연산 결과를 출력합니다. bool2와 bool4 모두 False이므로 or 연산 결과는 False입니다. ◆ 5

bool1 반대 논리값을 출력합니다. bool1이 True이므로 not bool1은 False입니다. 마찬가지로 not bool2는 bool2의 반대 논리값인 True가 됩니다. ◆ 6~7

초급

025

비트 연산자 이해하기 (&, |, ~, ^, >>, <<)

- 학습 내용 : 0과 1로 구성된 비트간 연산을 수행하는 비트 연산자에 대해 이해합니다.
- 힌트 내용 : 8비트를 1바이트라고 부릅니다.

소스 : 025.py

```
1: bit1 = 0x61
2: bit2 = 0x62
3: print(hex(bit1 & bit2))          # 0x600 | 출력됨
4: print(hex(bit1 | bit2))          # 0x630 | 출력됨
5: print(hex(bit1 ^ bit2))          # 0x3 이 출력됨
6: print(hex(bit1 >> 1))          # 0x300 | 출력됨
7: print(hex(bit1 << 2))          # 0x184가 출력됨
```

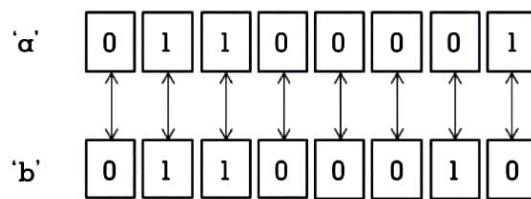
1비트는 0또는 1로만 표현될 수 있는 데이터 단위입니다. 1비트는 두 개의 값만 표시 가능하므로 1비트로 나타낼 수 있는 경우의 수는 두 가지입니다. 컴퓨터 분야에서 8비트는 1바이트로 부릅니다. 1바이트로 표현 가능한 경우의 수는 $2^8 = 256$ 가지입니다.

컴퓨터는 0과 1 두 개의 신호로 모든 연산과 데이터 처리를 수행합니다. 따라서 비트 연산은 컴퓨터 입장에서는 매우 기본적이고 근본적인 연산 방법이지만 사람 입장에서는 약간 생소하고 어려운 연산입니다.

다음의 예를 보면서 비트 연산에 대해 이해해봅니다.

문자 'a'는 컴퓨터 내부적으로 0110 0001로 표현하고 처리합니다. 마찬가지로 문자 'b'는 0110 0010으로 표현하고 처리합니다.

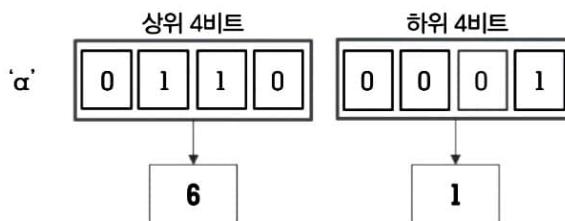
문자 'a'와 문자 'b'의 비트간 연산을 한다는 것은 'a'의 각 비트들과 'b'의 각 비트들을 자릿수에 맞게 비트별로 연산을 독립적으로 수행한다는 의미입니다.



파이썬에서 사용 가능한 비트 연산자는 다음과 같습니다.

비트 연산자	의미
<code>A & B</code>	A와 B의 비트간 and 연산을 수행함
<code>A B</code>	A와 B의 비트간 or 연산을 수행함
<code>A ^ B</code>	A와 B의 비트간 배타적 논리합 xor 연산을 수행함
<code>~A</code>	A의 비트를 반전시킴. 즉 A의 1의 보수를 만듦
<code>A >> n</code>	A의 모든 비트를 n만큼 오른쪽으로 시프트 시킴
<code>A << n</code>	A의 모든 비트를 n만큼 왼쪽으로 시프트 시킴

1바이트를 2진수로 표현하면 8자리 숫자가 되어 읽기에는 조금 깁니다. 이런 이유로 편의상 1바이트를 표현할 때 두 자리 16진수로 표현합니다. 1바이트를 이루는 8비트에서 왼쪽 4비트를 상위 4비트라 부르고 오른쪽 4비트를 하위 4비트라 부릅니다. 상위 4비트를 하나의 16진수로 대응시키고 하위 4비트를 또 하나의 16진수로 대응시켜 나타냅니다. 문자 'a'를 16진수로 나타내면 다음과 같이 '61'로 표현됩니다.



예제는 0x61과 0x62 두 개의 16진수를 이용해 여러 가지 비트 연산을 수행해보는 코드입니다.

- 1~2 ◆ bit1과 bit2에 각각 16진수 61, 62를 대입합니다. 16진수 61과 62는 2진수로 각각 0110 0001, 0110 0010이 됩니다.
- 3 ◆ 0110 0001 & 0110 0010을 연산하면 0110 0000이 됩니다. 이는 16진수로 60이 됩니다.
- 4 ◆ 0110 0001 | 0110 0010을 연산하면 0110 0011이 됩니다. 이는 16진수로 63이 됩니다.
- 5 ◆ 0110 0001 ^ 0110 0010을 연산하면 0000 0011이 됩니다. 이는 16진수로 3이 됩니다. 배타적 논리 합 XOR은 두 개의 비트값이 다른 경우에는 1, 같은 경우에는 0이 됩니다.
- 6 ◆ 0110 0001을 오른쪽으로 1만큼 시프트합니다. 오른쪽으로 시프트하는 경우에는 오른쪽 비트는 없어지고 왼쪽은 최상위 비트로 채워집니다. 결과는 0011 0000이 됩니다. 이는 16진수 값으로 30입니다. 만약 1011 0011 >> 1을 계산하면 시프트하기 전의 원래 값의 최상위 비트가 1이므로 시프트 한 결과는 1101 1001이 됩니다.
- 7 ◆ 0110 0001을 왼쪽으로 2만큼 시프트합니다. 왼쪽으로 시프트하는 경우에는 1이 시작되기 전까지 왼쪽 비트는 없어지고 오른쪽 비트는 0으로 채워집니다. 즉, 이동하는 왼쪽 비트가 1인 경우에는 그 값이 보존됩니다. 결과는 0001 1000 0100이 됩니다. 이는 16진수 값으로 184입니다.

비트 반전 연산자인 ~은 해당 값의 1의 보수로 비트를 반전시키지만 그 출력값을 이해하려면 2의 보수개념을 알아야 하는데 이는 이 책의 범위를 넘기 때문에 생략합니다.

시퀀스 자료형 이해하기

초급

026



- **학습 내용 :** 객체가 순서를 가지고 나열되어 있는 시퀀스 자료형을 이해하고, 시퀀스 자료형이 가지는 공통적인 특성을 배웁니다.
- **힌트 내용 :** 문자열 'abcde'는 문자 a, b, c, d, e가 순서대로 나열되어 있는 시퀀스 자료형입니다.

```

1: strdata = 'abcde'          # 문자열은 시퀀스 자료형임
2: listdata = [1, [2, 3], '안녕']  # 리스트는 시퀀스 자료형임
3: tupledata = (100, 200, 300)    # 튜플은 시퀀스 자료형임
  
```

시퀀스 자료형은 어떤 객체가 순서를 가지고 나열되어 있는 것을 말합니다. 예를 들어, 문자열 'abcde'는 문자 a, b, c, d, e가 순서를 가지고 차례대로 나열되어 있는 것입니다.

파이썬의 시퀀스 자료형에는 다음과 같은 것들이 있습니다.

- 문자열
- 리스트
- 튜플

문자열은 문자나 기호들이 순서대로 나열되어 있는 시퀀스 자료형입니다. strdata는 알파벳 'a', 'b', 'c', 'd', 'e'가 순서대로 나열되어 있는 문자열을 담고 있는 변수입니다. ◆ 1

리스트는 임의의 객체가 순서대로 나열되어 있는 시퀀스 자료형입니다. listdata는 1, [2, 3], '안녕'이 순서대로 나열되어 있는 리스트를 담은 변수입니다. ◆ 2

튜플은 리스트와 마찬가지로 값을 변경할 수 없는 임의의 객체가 나열되어 있는 시퀀스 자료형입니다. ◆ 3

파이썬의 시퀀스 자료형은 다음과 같은 공통적인 특성을 가지고 있습니다. 이 부분은 자주 사용되는 시퀀스 자료형의 특성이므로 잘 알아두어야 합니다.

특성	설명
인덱싱	인덱스를 통해 해당 값에 접근할 수 있습니다. 인덱스는 0부터 시작합니다.
슬라이싱	특정 구간의 값을 취할 수 있습니다. 구간은 시작 인덱스와 끝 인덱스로 정의합니다.
연결	'+' 연산자를 이용해 두 시퀀스 자료를 연결하여 새로운 시퀀스 자료로 생성합니다.
반복	'*' 연산자를 이용해 시퀀스 자료를 여러 번 반복하여 새로운 시퀀스 자료로 생성합니다.
멤버체크	'in' 키워드를 사용하여 특정 값이 시퀀스 자료의 요소로 속해 있는지 확인할 수 있습니다.
크기정보	<code>len()</code> 을 이용해 시퀀스 자료의 크기를 알 수 있습니다. 시퀀스 자료의 크기는 문자열의 경우 문자의 개수, 리스트와 튜플인 경우 멤버의 개수가 됩니다.

각 특성에 대한 자세한 내용은 예제 027~예제 032를 참고합니다.

시퀀스 자료 인덱싱 이해하기

초급

027

- 학습 내용 : 시퀀스 자료에서 인덱스를 이용해 멤버를 취하는 방법을 이해합니다.
- 힌트 내용 : 시퀀스 자료의 인덱스는 0부터 시작하며, 음수 인덱스도 사용 가능합니다.



소스 : 027.py

```

1: strdata = "Time is money!!"
2: listdata = [1, 2, [1, 2, 3]]
3: print(strdata[5])           # 'i'가 출력됨
4: print(strdata[-2])         # '!'가 출력됨
5: print(listdata[0])          # 1이 출력됨
6: print(listdata[-1])         # [1, 2, 3]이 출력됨
7: print(listdata[2][-1])       # 3이 출력됨

```

인덱싱(indexing)이란 시퀀스 자료형에서 인덱스를 통해 해당하는 값을 얻는 방법입니다. 파이썬에서 인덱스는 0부터 시작하며, 음수인 인덱스도 사용 가능합니다. 음수 인덱스는 우리가 일상에서 말하는 “끝에서부터 몇 번째..”라는 의미와 같습니다. 예제의 `strdata` 각 요소에 대해 인덱스를 표시하면 다음 표와 같습니다.

strdata	T	i	m	e		i	s		m	o	n	e	y	!	!
인덱스	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

이 표를 참고하여 예제 코드를 이해합니다.

- 3 ◆ strdata에서 인덱스가 5인 요소를 출력합니다. 인덱스는 0부터 시작하므로 strdata의 6번째 요소를 취합니다. strdata의 인덱스가 5인 요소는 'i'입니다.
- 4 ◆ strdata의 끝에서 두 번째 요소를 출력합니다. 해당 요소는 '!'입니다.
- 5 ◆ listdata에서 인덱스가 0인 요소를 출력합니다. 해당 요소는 1입니다.
- 6 ◆ listdata의 끝에서 첫 번째 요소, 즉 마지막 요소를 출력합니다. listdata의 마지막 요소는 [1, 2, 3]입니다.
- 7 ◆ listdata에서 인덱스가 2인 요소는 [1, 2, 3]입니다. listdata[2] = [1, 2, 3][0]으로 listdata[2][-1]은 listdata[2]의 마지막 요소를 나타내므로, [1, 2, 3]의 마지막 요소는 3입니다. 따라서 listdata[2][-1]의 값은 3입니다.

시퀀스 자료 슬라이싱 이해하기

초급

028

- 학습 내용 : 시퀀스 자료의 일부 범위를 취하는 슬라이싱에 대해 이해합니다.
- 힌트 내용 : 슬라이싱은 [시작 인덱스:끝 인덱스:스텝]으로 표현하며, 스텝은 생략해도 됩니다.



소스 : 028.py

```

1: strdata = 'Time is money!!'
2: print(strdata[1:5])           # 'ime'가 출력됨
3: print(strdata[:7])           # 'Time is'가 출력됨
4: print(strdata[9:])           # 'oney!!'가 출력됨
5: print(strdata[:-3])          # 'Time is mone'이 출력됨
6: print(strdata[-3:])          # 'y!!'이 출력됨
7: print(strdata[:])            # 'Time is money!!'가 출력됨
8: print(strdata[::-2])          # 'Tm smny!'가 출력됨

```

인덱싱은 인덱스에 해당하는 요소 하나를 취하는 방법이지만 슬라이싱은 시퀀스 자료에서 일정 범위에 해당하는 부분을 취하는 방법이며 표현은 다음과 같습니다.

[시작 인덱스:끝 인덱스:스텝]

- **시작 인덱스**: 슬라이싱 범위의 시작
- **끝 인덱스**: 슬라이싱 범위의 끝
- **스텝**: 자료를 취하는 간격

여기서 스텝은 생략해도 되며 디폴트 값은 1입니다. 따라서 보통 슬라이싱 범위는 [시작 인덱스:끝 인덱스]로 지정하며, 시작 인덱스 이상부터 끝 인덱스 미만까지 자료를 취하라는 뜻입니다. 이를 부등식으로 표현하면 다음과 같습니다.

시작 인덱스 \leq [시작 인덱스:끝 인덱스] < 끝 인덱스

다음 표는 슬라이싱 범위를 표현하는 다양한 방법입니다.

슬라이싱 범위	의미
[m:n]	시퀀스 자료의 인덱스가 m 이상 n 미만인 요소를 슬라이싱합니다.
[::n]	시퀀스 자료의 처음부터 인덱스가 n 미만인 요소까지 슬라이싱합니다.
[m:]	시퀀스 자료의 인덱스가 m인 요소부터 시퀀스 자료의 끝까지 슬라이싱합니다.
[:-n]	시퀀스 자료의 처음부터 끝에서 n번째 미만인 요소까지 슬라이싱합니다.
[~-m:]	시퀀스 자료의 끝에서 m번째 요소부터 시퀀스 자료의 끝까지 슬라이싱합니다.

예제 027에서 보인 시퀀스 자료와 인덱스와의 관계표를 참고하여 슬라이싱을 이해합니다.

- 2 ◆ `strdata[1:5]`은 인덱스가 1 이상이고 5 미만인 부분을 슬라이싱 합니다. 슬라이싱 범위는 'ime'입니다.
- 3 ◆ `strdata[:7]`은 처음부터 인덱스가 7 미만인 부분을 슬라이싱 합니다. 슬라이싱 범위는 'Time is'입니다.
- 4 ◆ `strdata[9:]`은 인덱스가 9 이상인 모든 요소를 슬라이싱 합니다. 슬라이싱 범위는 'oney!!'입니다.
- 5 ◆ `strdata[:-3]`은 처음부터 끝에서 3번째 미만인 요소까지 슬라이싱 합니다. 슬라이싱 범위는 'Time is mone'입니다.
- 6 ◆ `strdata[-3:]`은 끝에서 3번째 이상인 모든 요소를 슬라이싱 합니다. 슬라이싱 범위는 'y!!'입니다.
- 7 ◆ `strdata[:]`은 처음부터 끝까지 슬라이싱 하는 것이므로 `strdata` 자체가 됩니다.
- 8 ◆ 8라인은 스텝을 지정한 슬라이싱의 예입니다. `strdata[::-2]`는 처음부터 끝까지 스텝을 2로 해서 슬라이싱 하라는 의미입니다. 따라서 슬라이싱 되는 인덱스는 1, 3, 5, 7, 9, 11, 13이며, 이 부분에 해당하는 요소를 취하면 'Tm smny!'가 됩니다.

시퀀스 자료 연결 이해하기(+)

초급

029

- 학습 내용 : 자료형이 같은 시퀀스 자료 두 개를 순서있게 연결하는 방법을 이해합니다.
- 힌트 내용 : 시퀀스 자료를 연결하는 연산자는 '+'입니다.



소스 : 029.py

```

1: strdata1 = 'I love'; strdata2 = 'Python'; strdata3 = 'you'
2: listdata1 = [1, 2, 3]; listdata2 = [4, 5, 6]
3: print(strdata1 + strdata2)           # 'I love Python'이 출력됨
4: print(strdata1 + strdata3)           # 'I love you'가 출력됨
5: print(listdata1 + listdata2)         # [1, 2, 3, 4, 5, 6]이 출력됨
  
```

자료형이 동일한 두 개의 시퀀스 자료는 + 연산자로 순서있게 연결하여 새로운 시퀀스 자료로 만들 수 있습니다. 다시 말해 문자열 + 문자열, 리스트 + 리스트, 튜플 + 튜플과 같이 두 개의 동일한 시퀀스 자료형에 대해 '+' 연산자로 연결 가능하다는 말이며, 문자열과 리스트를 '+'로 연결할 수는 없습니다.

세 개의 문자열 strdata1, strdata2, strdata3을 정의하고, 두 개의 리스트 listdata1, listdata2를 정의합니다.

◆ 1-2

문자열 strdata1과 strdata2를 '+'로 연결한 새로운 문자열과 strdata1과 strdata3을 '+'로 연결한 새로운 문자열을 출력합니다.

◆ 3-4

리스트인 listdata1과 listdata2를 '+'로 연결한 새로운 리스트를 만들고 출력합니다.

◆ 5

초급

030

시퀀스 자료 반복 이해하기(*)

- 학습 내용 : 동일한 시퀀스 자료를 반복 연결하는 방법에 대해 이해합니다.
- 힌트 내용 : 동일한 시퀀스 자료의 반복을 위한 연산자는 '*'입니다.

소스 : 030.py

```
1: artist = '빅뱅'
2: sing = '뱅~'
3: dispdata = artist + '이 부르는 ' + sing*3
4: print(dispdata)          # '빅뱅이 부르는 뱅~뱅~뱅~'이 출력됨
```

동일한 시퀀스 자료를 반복하여 새로운 시퀀스 자료를 만들고자 하면 '*' 연산자를 사용합니다. 시퀀스 자료 seqdata를 n번 반복하여 새로운 자료로 만드려면 seqdata*n이 됩니다.

3 ◆ dispdata는 artist + '이 부르는 ' + sing*3이 연결된 문자열입니다. sing*3은 '뱅~'*3이므로 '뱅~'을 3번 반복한 '뱅~뱅~뱅~'이 됩니다. 따라서 예제 코드의 결과는 '빅뱅이 부르는 뱅~뱅~뱅~'입니다.

만약 리스트 [1, 2, 3]을 5번 반복하기 위해 [1, 2, 3]*5를 수행하면 결과는 [1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]이 됩니다.

시퀀스 자료 크기 이해하기(len)

초급

031

- 학습 내용 : 시퀀스 자료의 크기에 대해 이해하고 자료의 크기를 구하는 방법을 이해합니다.
- 힌트 내용 : 모든 자료의 크기나 길이를 구하는 함수는 len()입니다.

 소스 : 031.py

```

1: strdata1 = 'I love python'
2: strdata2 = '나는 파이썬을 사랑합니다'
3: listdata = ['a', 'b', 'c', strdata1, strdata2]
4: print(len(strdata1))           # 130 | 출력됨
5: print(len(strdata2))           # 130 | 출력됨
6: print(len(listdata))           # 5가 출력됨
7: print(len(listdata[3]))         # 130 | 출력됨

```

모든 시퀀스 자료는 고정된 길이 또는 크기를 가지고 있습니다. 시퀀스 자료의 크기는 시퀀스 자료를 구성하는 요소의 개수입니다. 문자열은 문자가 요소이고, 리스트와 튜플은 임의의 객체가 요소이며 콤마(,)로 구분되어 있습니다. 따라서 문자열의 크기는 문자열을 구성하는 문자의 개수이며, 리스트와 튜플의 크기는 콤마로 구분되어 있는 요소의 개수입니다.

strdata1은 공백을 포함하여 모두 13자로 구성된 문자열 'I love python'을 담은 변수입니다. 따라서 len(strdata1)은 13입니다. strdata2는 공백을 포함하여 한글로 구성된 문자열이며 모두 13자입니다. 따라서 len(strdata2)는 13입니다.

◆ 4-5

listdata는 5개의 요소를 가진 리스트 ['a', 'b', 'c', strdata1, strdata2]를 담은 변수입니다. 따라서 len(listdata)는 5가 됩니다.

◆ 6

listdata[3]의 값은 strdata1이므로 len(listdata[3])은 13입니다.

◆ 7

초급

032

멤버체크 이해하기(in)



- 학습 내용 : 어떤 값이 시퀀스 자료에 있는지 없는지 확인하는 방법을 배웁니다.

- 힌트 내용 : in은 시퀀스 자료의 요소를 확인할 때 사용되는 키워드입니다.

소스 : 032.py

```
1: listdata = [1, 2, 3, 4]
2: ret1 = 5 in listdata      # False
3: ret2 = 4 in listdata      # True
4: print(ret1); print(ret2)
5: strdata = 'abcde'
6: ret3 = 'c' in strdata      # True
7: ret4 = 'l' in strdata      # False
8: print(ret3); print(ret4)
```

in은 자료에 어떤 값이 있는지 없는지 확인할 때 사용하는 키워드입니다. 활용하는 방법은 다음과 같습니다.

<값> in <자료>

값이 자료에 있으면 결과는 True이고, 없으면 False가 됩니다.

- 1~4 ◆ listdata에 5가 있는지 체크하고 그 결과를 ret1에 대입합니다. listdata에 4가 있는지 체크하고 그 결과를 ret2에 대입합니다. listdata에 5는 없고, 4는 있으므로 ret1, ret2는 각각 False, True가 됩니다.
- 5~8 ◆ strdata에 문자 'c'가 있는지 체크하고 그 결과를 ret3에 대입합니다. strdata에 문자 'l'이 있는지 체크하고 그 결과를 ret4에 대입합니다. strdata에 문자 'c'는 있고 문자 'l'은 없으므로 ret3, ret4는 각각 True, False가 됩니다.

문자열 이해하기

초급

033

- 학습 내용 : 문자열을 정의하고 선언하는 방법에 대해 살펴봅니다.
- 힌트 내용 : 문자열을 나타내는 방법은 '' "", """ """" 가 있습니다.



```

1: strdata1 = '나는 파이썬 프로그래머다'
2: strdata2 = "You are a programmer"
3: strdata3 = """I love
4:     python. You love
5: python too!
6: """
7: strdata4 = "My son's name is John"
8: strdata5 = '문자열 "abc"의 길이는 3입니다.'
# 문자열 선언 방법 1
# 문자열 선언 방법 2
# 문자열 선언 방법 3
# " " 안에서 ' 사용하기
# '' 안에서 " 사용하기

```

문자열은 문자나 기호가 순서로 나열되어 있는 시퀀스 자료입니다. 문자열을 선언하는 방법은 세 가지가 있습니다.

strdata1은 ''를 이용해 '나는 파이썬 프로그래머다'라는 문자열을 정의합니다. ◆ 1

strdata2는 " "를 이용해 "You are a programmer"라는 문자열을 정의합니다. ◆ 2

strdata3은 """ """를 이용해 문자열을 정의합니다. """ """로 둘러싸인 부분이면 줄바꿈이 되더라도 모든 것이 문자열로 정의됩니다. """ """은 블록 단위로 주석 처리할 때도 사용됩니다. ◆ 4~6

My son's name is John과 같이 문자열에 '이 포함되어 있으면 " "를 이용해 선언하면 됩니다. ◆ 7

문자열 "abc"의 길이는 3입니다와 같이 문자열에 "이 포함되어 있으면 ''를 이용해 선언하면 됩니다. ◆ 8

만약 "와 '이 섞여 있는 경우 """ """를 이용하거나 이스케이프 문자를 이용하면 됩니다. 이스케이프 문자는 예제 035에서 다룹니다.

초급

034



문자열 포맷팅 이해하기

- 학습 내용 : 값이 변하는 문자열을 표현하기 위한 문자열 포맷팅에 대해 이해합니다.
- 힌트 내용 : 문자열에서 변하는 값을 나타내는 포맷 문자열은 % 기호를 이용합니다.

소스 : 034.py

```
1: txt1 = '자바'; txt2 = '파이썬'
2: num1 = 5; num2 = 10
3: print('나는 %s보다 %s에 더 익숙합니다.' %(txt1, txt2))
4: print('%s은 %s보다 %d배 더 쉽습니다.' %(txt2, txt1, num1))
5: print('%d + %d = %d' %(num1, num2, num1+num2))
6: print('작년 세계 경제 성장률은 전년에 비해 %d%% 포인트 증가했다.' %num1)
```

문자열 포맷팅이란 변하는 값을 포함하는 문자열을 표현하기 위해 하나의 양식으로 문자열을 만드는 것입니다. 문자열 포맷팅에서 변하는 값을 나타내기 위해 사용되는 기호를 **포맷 문자열**이라고 하며, 자주 사용하는 포맷 문자열은 다음과 같습니다

포맷 문자열	설명
%s	문자열에 대응됨
%c	문자나 기호 한 개에 대응됨
%f	실수에 대응됨
%d	정수에 대응됨
%%	'%'라는 기호 자체를 표시함

- 3 ◆ '나는 %s보다 %s에 더 익숙합니다.'에는 두 개의 포맷 문자열 %s가 있습니다. 이 부분에 실제적인 값을 대입하려면 %(첫 번째 %s에 대입할 문자열, 두 번째 %에 대입할 문자열)을 뒤에 추가합니다. 3라인은 첫 번째 %s에 txt1을, 두 번째 %s에 txt2를 대입하라는 것이므로, '나는 자바보다 파이썬에 더 익숙합니다.'가 출력됩니다.
- 4 ◆ 포맷 문자열 %s %s %d에 각각 txt2, txt1, num1이 대입되어 '파이썬은 자바보다 5배 더 쉽습니다.' 가 출력됩니다.

◆ 5 포맷 문자열 %d %d %d에 각각 num1, num2, num1+num2가 대입되어 '5 + 10 = 15'가 출력됩니다.

◆ 6 포맷 문자열 %d%%에서 %%는 기호 '%' 자체를 의미하므로 '작년 세계 경제 성장률은 전년에 비해 5% 포인트 증가했다'가 출력됩니다.



N O T E | 변하는 값을 화면의 같은 라인에 출력하는 방법 |

파일을 다운로드 받거나 복사를 하는 경우 진행 상황을 사용자에게 피드백하는 가장 일반적인 방법은 GUI 환경에서는 프로그레스바로, 명령 프롬프트에서는 %로 진행률을 표시해주는 것입니다. 다음 소스 코드는 명령 프롬프트에서 진행률을 화면의 동일한 라인에 표시하는 방법을 제시한 것입니다.

소스 : 034-1.py

```
from time import sleep
for i in range(100):
    msg = '\r진행률 %d%%' %(i+1)
    print(' '*len(msg), end="")
    print(msg, end="")
    sleep(0.1)
```

이 코드에서 핵심은 msg='\r진행률 %d%%' %(i+1)과 print(' '*len(msg), end="") 부분입니다.

msg를 캐리지 리턴 문자 '\r'을 추가하여 문자열 포맷팅을 이용하여 구성하고 print(' '*len(msg), end="")로 msg 길이만큼 공백문자를 줄바꿈 없이 화면에 출력합니다. 그런 후 print(msg, end='')로 msg를 출력합니다. 캐리지 리턴은 현재 위치를 나타내는 커서를 화면 맨 앞으로 이동하라는 의미입니다. 윈도우 명령 프롬프트에서 위 소스 코드가 저장된 파일을 파이썬으로 실행시켜 봅니다.

초급

035



이스케이프 문자 이해하기

- 학습 내용 : 키보드로 입력하기 어려운 기호 등을 나타내기 위한 이스케이프 문자를 이해합니다.
- 힌트 내용 : 이스케이프 문자는 '\'로 시작합니다.

소스 : 035.py

```
1: print('나는 파이썬을 사랑합니다.\n파이썬은 자바보다 훨씬 쉽습니다.')
2: print('Name: John Smith\tSex: Male\tAge: 22')
3: print('이 문장은 화면폭에 비해 너무 길어 보기가 힘듭니다.\')
4: 그래서 \\Enter키를 이용해 문장을 다음줄과 연속되도록 했습니다.')
5: print('작은따옴표(')와 큰 따옴표(")는 문자열을 정의할 때 사용합니다.'
```

이스케이프 문자는 키보드로 입력하기 어려운 기호를 나타내기 위해 역슬래쉬 '\'로 시작하는 문자입니다. 그리고 인용부호(' 또는 " ") 안에 동일한 인용부호를 입력하는 경우에도 이스케이프 문자가 사용됩니다. 파이썬에서 자주 사용되는 이스케이프 문자는 다음과 같습니다.

이스케이프 문자	설명
\n	줄 바꾸기
\t	탭
\nEnter	줄 계속 (다음 줄도 계속되는 줄이라는 표시)
\\"	'\' 기호 자체
' 또는 "	' 기호 또는 " 기호 자체

예제 코드를 실행하면 다음과 같이 출력됩니다.



결과

나는 파이썬을 사랑합니다.

파이썬은 자바보다 훨씬 쉽습니다.

Name: John Smith Sex: Male Age: 22

이 문장은 화면폭에 비해 너무 길어 보기 가 힘듭니다. 그래서 \Enter키를 이용해 문장을 다음줄과
연속되도록 했습니다.

작은따옴표(')와 큰 따옴표(")는 문자열을 정의할 때 사용합니다.

초급

036



리스트 이해하기([])

- 학습 내용 : 순서를 갖는 임의의 객체 집합인 리스트에 대해 이해합니다.
- 힌트 내용 : 리스트는 임의의 객체를 요소로 가질 수 있으며, []로 표시합니다.

소스 : 036.py

```
1: list1 = [1, 2, 3, 4, 5]
2: list2 = ['a', 'b', 'c']
3: list3 = [1, 'a', 'abc', [1, 2, 3, 4, 5], ['a', 'b', 'c']]
4: list1[0] = 6
5: print(list1)          # [6, 2, 3, 4, 5]가 출력됨
6: def myfunc():
7:     print('안녕하세요')
8: list4 = [1, 2, myfunc]
9: list4[2]()            # '안녕하세요' 가 출력됨
```

리스트는 파이썬에서 가장 많이 활용되는 시퀀스 자료형 중 하나입니다. 리스트는 []로 표시하며 [] 안에 요소를 콤마로 구분하여 순서있게 나열합니다. 리스트의 요소로는 임의의 객체가 가능합니다. 임의의 객체란 숫자, 문자, 문자열, 리스트, 튜플, 사전 등 파이썬 자료형과 함수, 클래스 등을 말합니다.

- 1 ◆ list1은 정수 1에서 5까지 5개의 요소를 가지는 리스트입니다.
- 2 ◆ list2는 문자 'a', 'b', 'c' 3개의 요소를 가지는 리스트입니다.
- 3 ◆ list3은 정수 1, 문자 'a', 문자열 'abc', 리스트 [1, 2, 3, 4, 5], 리스트 ['a', 'b', 'c'] 5개의 요소를 가지는 리스트입니다.
- 4-5 ◆ 튜플과 달리 리스트의 요소는 값이 변경 가능합니다. list1[0]에 6을 대입하고 list1을 출력하면 list1의 첫 번째 요소가 6으로 변경되어 [6, 2, 3, 4, 5]가 표시됩니다.
- 6-9 ◆ myfunc()라는 함수를 정의합니다. 이 함수를 호출하면 단순히 '안녕하세요'를 출력합니다. list4는 정수 1, 2, 함수 myfunc를 요소로 가지는 리스트입니다. list4[2]는 myfunc이므로 list4[2]()는 myfunc()를 실행하는 것과 같으므로 '안녕하세요'가 출력됩니다.

튜플 이해하기(())

초급

037

- 학습 내용 : 리스트와 비슷하나 요소의 값을 변경할 수 없는 튜플에 대해 이해합니다.
- 힌트 내용 : 튜플은 ()로 표시하며 튜플의 요소를 변경하려고 하면 오류가 발생합니다.



소스 : 037.py

```

1: tuple1 = (1, 2, 3, 4, 5)
2: tuple2 = ('a', 'b', 'c')
3: tuple3 = (1, 'a', 'abc', [1, 2, 3, 4, 5], ['a', 'b', 'c'])
4: tuple1[0] = 6
5:
6: def myfunc():
7:     print('안녕하세요')
8:
9: tuple4 = (1, 2, myfunc)
10: tuple4[2]()
# '안녕하세요' 가 출력됨

```

튜플은 리스트와 비슷한 성질을 가지고 있는 자료형이지만 요소의 값을 변경할 수 없다는 것이 특징입니다.

튜플은 리스트와 마찬가지로 임의의 객체를 요소로 가질 수 있는 시퀀스 자료형입니다. ◆ 1-3

`tuple1[0]`의 값을 6으로 대입하여 변경하고자 하면 다음과 같은 오류가 발생합니다. ◆ 4

`TypeError: 'tuple' object does not support item assignment`

이는 튜플의 요소는 그 값을 변경할 수 없기 때문입니다.

오류가 발생한 4라인을 삭제하거나 주석 처리하고 코드를 실행하면 10라인에서 `myfunc()`가 실행되어 '안녕하세요'가 출력됩니다. ◆ 9-10

초급

038

사전 이해하기({ })

- 학습 내용 : 키와 값을 하나의 요소로 가지는 사전 자료에 대해 이해합니다.
- 힌트 내용 : 사전은 { }로 표시하며 요소들 사이의 순서가 없습니다.

소스 : 038.py

```
1: dict1 = {'a':1, 'b':2, 'c':3}
2: print(dict1['a'])          # 1이 출력됨
3: dict1['d'] = 4
4: print(dict1)              # {'a':1, 'b':2, 'c':3, 'd':4}가 출력되나 순서가 틀릴 수 있음
5: dict1['b'] = 7
6: print(dict1)              # {'a':1, 'b':7, 'c':3, 'd':4}가 출력되나 순서가 틀릴 수 있음
7: print(len(dict1))        # 4가 출력됨
```

사전은 키와 값을 하나의 요소로 하는 순서가 없는 집합입니다. 그러므로 사전은 시퀀스 자료형이 아니며 인덱싱으로 값을 접근할 수도 없습니다. 사전의 키와 값은 임의의 객체가 될 수 있습니다. 사전은 키:값 쌍이 하나의 요소입니다.

- ◆ 요소가 'a':1, 'b':2, 'c':3인 사전 dict1을 선언합니다. dict1은 3개의 키:값 쌍으로 구성되어 있습니다.
- ◆ 사전은 시퀀스 자료가 아니므로 dict1[0]과 같이 인덱싱으로 값을 얻을 수 없고, 키로 접근해야 합니다. dict1['a']는 dict1의 요소 중 키가 'a'인 값을 의미합니다. 따라서 1이 출력됩니다.
- ◆ dict1에 새로운 키:값을 추가하는 방법입니다. dict1['d'] = 4는 dict1에 키가 'd', 값이 4인 요소를 추가라는 의미입니다.
- ◆ 만약 동일한 키가 있다면 키에 대응하는 값을 변경하게 됩니다. dict1['b'] = 7은 키가 'b'인 요소가 존재하므로 해당 요소의 값을 7로 변경합니다.
- ◆ 사전의 크기를 구할 때는 시퀀스 자료형의 크기를 구할 때 사용했던 len()을 사용하면 됩니다. 사전의 크기는 콤마로 구분되는 요소들의 개수입니다.

함수 이해하기(def)

초급

039

- 학습 내용 : 특정 목적을 가진 독립된 코드 집합인 함수에 대해 이해합니다.
- 힌트 내용 : 파이썬에서 함수는 def로 시작하여 정의합니다.



소스 : 039.py

```

1: def add_number(n1, n2):
2:     ret = n1+n2
3:     return ret
4:
5: def add_txt(t1, t2):
6:     print(t1+t2)
7:
8: ans = add_number(10, 15)
9: print(ans)                      # 25가 출력됨
10: text1 = '대한민국~'
11: text2 ='만세!!!'
12: add_txt(text1, text2)          # '대한민국~만세!!!'가 출력됨

```

함수란 특정 목적을 가진 코드의 집합이며 독립적으로 호출될 수 있는 것을 말합니다.

예를 들어, 두 수를 입력받아 더한 값을 계산하는 코드가 필요하다고 생각해 봅니다. 두 수를 더하는 기능이 자신이 만들고자 하는 프로그램 여러 곳에서 사용된다고 할 때 이 부분만을 따로 구성하여 재사용할 수 있도록 하면 효율적인 프로그래밍이 될 것입니다. 이런 경우 두 수를 더하여 그 값을 돌려주는 기능을 하는 코드만을 따로 모아 함수로 구성하면 됩니다.

파이썬에서 함수를 정의하는 방법은 다음과 같습니다.

인자와 리턴값이 있는 함수 선언 방법

```

def 함수이름(인자1, 인자2, ...):
    코드들
    return 결과값

```

정의하고자 하는 함수에 인자가 필요없다면 다음과 같이 정의합니다.

리턴값은 있지만 인자가 없는 함수 선언 방법

```
def 함수이름():  
    코드들  
    return 결과값
```

함수의 리턴값이 없는 경우에는 `return`만 하거나 `return`을 생략하면 됩니다.

인자와 리턴값이 없는 함수 선언 방법

```
def 함수이름():  
    코드들  
    return (또는 생략)
```

정의된 함수를 호출하는 방법은 매우 쉽습니다. 함수 인자 자리에 실제값을 대입하여 함수이름만 호출하면 됩니다. 함수의 리턴값이 있는 경우에는 다음과 같이 함수 리턴값을 받을 변수를 저장해 주면 됩니다. 함수의 인자가 없는 경우에는 함수이름만 호출하면 됩니다.

인자와 리턴값이 있는 함수 호출 방법

```
변수 = 함수이름(값1, 값2, ...)
```

리턴값이 없는 함수 호출 방법

```
함수이름(값1, 값2, ...)
```

인자와 리턴값이 없는 함수 호출 방법

```
함수이름()
```

함수는 재사용 목적이 아니더라도 복잡한 알고리즘을 독립적인 영역에 따로 구현함으로써 소스 코드의 가독성을 높일 수 있게 해줍니다.

함수이름이 `add_number`이고 인자가 `n1, n2`인 함수를 정의합니다. 이 함수는 인자 `n1, n2`를 더한 값을 리턴합니다. ◆ 1-3

함수이름이 `add_txt`이고 인자가 `t1, t2`인 함수를 정의합니다. 이 함수는 인자 `t1, t2`를 더한 값을 출력하며, 리턴값은 없습니다. ◆ 5-6

`add_number` 함수를 호출하는데 인자 자리에 10과 15를 대입했습니다. `add_number` 함수는 인자로 입력된 10과 15를 더하고 이 값을 리턴하며, 리턴값을 `ans`로 둡니다. 화면에 25가 출력됩니다. ◆ 8-9

`add_txt` 함수를 호출하는데 인자 자리에 `text1, text2`를 대입했습니다. `text1`과 `text2`의 실제 값은 ‘대한민국~’과 ‘만세!!’입니다. `add_txt` 함수는 인자로 입력된 이 두 문자열을 연결한 결과를 출력합니다. ◆ 10-12

`add_number`와 `add_txt` 함수에 다양한 수나 문자열을 인자로 입력해보고 어떤 결과가 나오는지 확인해 보기 바랍니다.

초급

040

함수 인자 이해하기



- 학습 내용 : 함수로 값을 전달하는 인자에 대해 이해합니다.
- 힌트 내용 : 함수에 값을 전달하기 위해 인자순서, 기본인자, 키워드인자, 가변인자, 미정 키워드인자를 이용하는 방법이 있습니다.

소스 : 040.py

```
1: def add_txt(t1, t2='파이썬'):  
2:     print(t1+' : ' +t2)  
3:  
4: add_txt('베스트')                      # '베스트' : 파이썬이 출력됨  
5: add_txt(t2='대한민국', t1='1등')        # '1등' : 대한민국이 출력됨  
6:  
7: def func1(*args):  
8:     print(args)  
9:  
10: def func2(width, height, **kwargs):  
11:    print(kwargs)  
12:  
13: func1()                                # 빈 튜플 ()이 출력됨  
14: func1(3, 5, 1, 5)                      # (3, 5, 1, 5)가 출력됨  
15: func2(10, 20)                          # 빈 사전 {}이 출력됨  
16: func2(10, 20, depth=50, color='blue')  # {'depth':50, 'color':'blue'} 이 출력됨
```

인자의 위치에 실제값을 대입하여 함수를 호출하면 인자 순서에 대응되는 값을 함수 코드에 대입하여 실행하게 됩니다. 예제 039에서 함수에 값을 전달하는 방법은 모두 이 원리에 따릅니다.

함수호출부분 answer = func(10, 20, 30):

함수선언부분 def func(p1, p2, p3):

위에서 설명한 함수에 값을 전달하는 기본적인 방법 이외에 다음과 같은 방법들이 있습니다.

- **기본 인자를 이용해 값을 전달하는 방법:** 기본 인자 이용
- **인자 이름을 이용해 값을 전달하는 방법:** 키워드 인자 이용
- **가변 인자 튜플을 이용해 값을 전달하는 방법:** 가변 인자 이용
- **미정 키워드 인자 사전을 이용해 값을 전달하는 방법:** 미정 키워드 인자 이용

함수에 값을 전달하는 이들 방법에 대해 예제를 통해 이해해봅니다.

`add_txt` 함수를 정의할 때 두 번째 인자인 `t2`에 기본값 '파이썬'을 지정했습니다. 이는 함수 호출 시 `t2` 위치에 실제값을 전달하지 않아도 '파이썬'이 기본값으로 대입됩니다. `add_txt`는 `t1`과 `t2`를 콜론(:)으로 연결한 뒤 출력하는 함수입니다.

◆ 1-2



기본값을 대입한 기본 인자는 마지막에 위치해야 합니다. 만약 다음과 같이 기본 인자의 위치를 일반 인자 앞에 두면 오류가 발생합니다.

```
def add_txt(t1='파이썬', t2):
    → SyntaxError: non-default argument follows default argument
```

`add_txt('베스트')`는 `add_txt` 함수의 첫 번째 인자인 `t1`에 '베스트'가 대입됩니다. `t2`에 아무런 값이 지정되지 않았으므로 기본값으로 지정된 '파이썬'이 `t2`에 적용됩니다.

◆ 4

`add_txt` 함수를 호출할 때 인자 이름을 명시하고 값을 대입했습니다. 이와 같이 인자 이름을 통해 함수로 값을 전달하는 방법을 '키워드 인자에 의한 값 전달 방법'이라 합니다. 키워드 인자를 이용하면 인자의 순서는 무시됩니다.

◆ 5

함수호출부분 `answer = func(p2=10, p1=20, p3=30):`

함수선언부분 `def func(p1, p2, p3):`

- 7-8 ◆ 때로는 함수를 정의할 때 인자의 개수가 불명확한 경우가 있을 수 있습니다. 이 경우 가변 인자를 활용하여 함수를 선언하면 됩니다. 가변 인자는 `*args`와 같이 인자 이름 앞에 `*`를 붙입니다. `args`는 함수 내부에서 튜플로 처리됩니다. `func1`은 가변 인자인 `args`를 출력하는 함수입니다.
- 10-11 ◆ 마찬가지로 키워드 인자가 불명확한 경우도 있을 수 있습니다. `func2`를 정의할 때 `width`와 `height`는 명확하지만, 차후 어떤 키워드 인자가 필요한지 확정되지 않은 경우 `**kwargs`와 같이 인자를 정해주면 됩니다. `kwargs`는 함수 내부에서 사전으로 처리됩니다. `func2`는 미정 키워드 인자 `kwargs`를 출력하는 함수입니다.
- 13-14 ◆ `func1`에 아무런 값을 전달하지 않았을 경우 `args`는 빈 튜플이 됩니다. 만약 3, 5, 1, 5와 같이 네 개의 값을 인자로 함수에 전달하면 `args`에 이 값들을 (3, 5, 1, 5)와 같이 튜플로 담아 함수 내부에서 처리합니다.
- 15-16 ◆ `func2`에 `width`와 `height`에 해당하는 값인 10, 20만 전달하여 호출하면 `**kwargs`에 전달되는 값이 없으며, `kwargs`는 함수 내부에서 빈 사전 자료가 됩니다. `func2`에 `width`, `height`외에 키워드 인자로 `depth=50, color='blue'`를 추가하여 호출하면 `**kwargs`는 `depth=50, color='blue'`를 전달받고, `func2` 함수 내부에서 `kwargs`는 {'`depth':50, 'color':'blue'}`와 같은 사전 자료가 됩니다.

지역변수와 전역변수 이해하기 (global)

초급

041

- 학습 내용 :** 지역변수와 전역변수의 차이를 이해합니다.
- 힌트 내용 :** 지역변수는 함수 내부에서만 유효하고 전역변수는 코드 전반에 걸쳐 유효합니다. 함수의 인자는 지역변수입니다.



소스 : 041.py

```

1: param = 10
2: strdata = '전역변수'
3:
4: def func1():
5:     strdata = '지역변수'
6:     print(strdata)
7:
8: def func2(param):
9:     param = 1
10:
11: def func3():
12:     global param
13:     param = 50
14:
15: func1()           # '지역변수'가 출력됨
16: print(strdata)    # '전역변수'가 출력됨
17: print(param)      # 100 출력됨
18: func2(param)
19: print(param)      # 100 출력됨
20: func3()
21: print(param)      # 500 출력됨

```

변수의 유효한 범위를 기준으로 구분할 때 지역변수와 전역변수라는 것이 있습니다. 지역변수는 함수 내에서만 유효한 변수이고, 전역변수는 코드 전반에 걸쳐 유효한 변수입니다. 따라서 지역변수를 선언하는 위치는 함수 내부가 되며, 전역변수를 선언하는 위치는 함수 바깥이 됩니다. 지역변수는 함수를 벗어나면 더 이상 유효하지 않습니다. 함수의 인자로 선언된 변수는 함수 내에서만 유효한 지역변수입니다.

만약 함수 내에서 선언한 지역변수와 함수 밖에서 선언한 전역변수가 이름이 같을 경우에는 어떻게 될까요? 함수 내에서 선언한 지역변수가 함수 바깥에서 선언된 전역변수와 이름이 같을 경우, 함수 내에서는 지역변수가 우선이기 때문에 지역변수로 취급되고 처리됩니다. 함수 내부에서 바깥에서 선언한 전역변수를 사용하려면 ‘global’ 키워드를 이용해 전역변수를 사용한다고 명시하면 됩니다.

- 1-2 ◆ 함수 바깥에서 `param`과 `strdata`를 각각 10, ‘전역변수’로 선언했습니다.
- 4-6 ◆ `func1()` 내에서 `strdata`에 ‘지역변수’라는 문자열을 대입했습니다. `strdata`는 함수 내부에서 선언되었으므로 지역변수가 되며, `func1()`을 벗어나면 더 이상 유효한 변수가 아닙니다.
- 8-9 ◆ `func2()`는 `param`이라는 이름의 인자를 한 개 가지고 있습니다. 인자이름 `param`은 전역변수와 이름이 같지만 `func2()` 내에서만 유효한 지역변수로 취급되고 처리됩니다. 따라서 `func2()` 내에서 `param`에 값을 대입해도 전역변수 `param`에 영향을 주지 않습니다.
- 11-13 ◆ `func3()` 내부에서 `global param`으로 전역변수로 선언된 `param`을 사용할 것임을 명시했습니다. 이제 `func3()` 내에서 `param`은 전역변수 `param`을 이용하는 것이며 이 값을 변경하게 되면 전역변수 `param`의 값이 변경되는 것입니다.
- 15-16 ◆ `func1()`을 호출하면 `func1()` 내에서 선언된 지역변수 `strdata`의 값인 ‘지역변수’를 출력합니다. 하지만 전역변수 `strdata`를 출력하면 ‘전역변수’가 출력됩니다.
- 17-19 ◆ 먼저 전역변수 `param`의 값을 출력해보면 10이 나옵니다. `func2()`의 인자로 전역변수 `param`을 대입하여 호출하면 `func2()`는 이를 자신의 인자인 `param`에 대입합니다. `func2()`의 인자 `param`은 지역변수로 처리되므로 `func2()` 내부에서 `param`의 값을 변경해도 전역변수 `param`에 영향을 주지 않습니다. `func2()` 호출 이후 전역변수 `param`의 값을 출력해보면 10으로 변동이 없습니다.
- 20-21 ◆ `func3()`는 전역변수 `param`의 값을 50으로 변경하는 함수입니다. 이 함수를 호출한 후 전역변수 `param`의 값을 출력해보면 50이 됩니다.

함수 리턴값 이해하기(return)

초급

042

- 학습 내용 : 함수가 수행되고 난 후 결과값 반환에 대한 내용을 이해합니다.
- 힌트 내용 : 함수는 한 개 이상의 값을 리턴할 수 있으며, 리턴값이 없을 수도 있습니다.



소스 : 042.py

```

1: def reverse(x, y, z):
2:     return z, y, x
3:
4: ret = reverse(1, 2, 3)
5: print(ret)           # (3, 2, 1)이 출력됨
6:
7: r1, r2, r3 = reverse('a', 'b', 'c')
8: print(r1); print(r2); print(r3)    # 'c', 'b', 'a' 순으로 출력됨

```

함수가 수행되고 나면 그 결과를 리턴하고 종료하는 경우가 많습니다. 이때 `return` 키워드를 이용하여 값을 리턴하고 함수를 종료합니다. 만약 리턴값이 없는 경우에는 `return` 없이 함수 코드를 마무리 해도 됩니다. 리턴값이 여러 개인 경우에는 튜플로 리턴값을 만들어 리턴합니다.

`reverse()`는 3개의 인자를 가지고 있으며, 이 함수는 인자의 순서를 바꾸어 리턴합니다. 따라서 `reverse()`의 리턴값은 3개입니다.

◆ 1-2

숫자 1, 2, 3을 `reverse()`의 인자로 전달하여 호출하고 그 결과값을 `ret`으로 둡니다. `ret`을 출력하면 튜플 (3, 2, 1)이 표시됩니다.

◆ 4-5

리턴값이 여러 개인 경우 튜플 형식으로 리턴값이 만들어지므로 튜플의 요소 개수만큼 나누어서 리턴값을 개별적으로 받을 수도 있습니다. `r1, r2, r3`에 각각 'c', 'b', 'a'가 리턴값으로 대입됩니다.

◆ 7-8

초급

043

파이썬 모듈 이해하기

- 학습 내용 : 파이썬 모듈을 이해하고 자신의 코드에 모듈을 포함시키는 방법을 배웁니다.
- 힌트 내용 : 파이썬 모듈은 보통 하나의 독립된 파이썬 소스 파일로 구성되어 있습니다.

소스 : 043.py

```
1: import time  
2:  
3: print('5초간 프로그램을 정지합니다.')  
4: time.sleep(5)  
5: print('5초가 지나갔습니다.')
```

코드를 작성할 때 이미 만들어져 있는 함수들을 활용하면 보다 효율적이고 빠르게 개발할 수 있습니다. 이미 만들어져 있고 안정성이 검증된 함수들을 성격에 맞게 하나의 파이썬 파일에 묶어 만들어 놓은 것을 **모듈**이라 부릅니다.

외부 모듈에 있는 함수들을 활용하려면 이 모듈을 먼저 우리 코드로 가져와서 자유롭게 사용할 수 있도록 해야하는데 이런 일을 파이썬에서는 **모듈을 임포트(import)한다고 합니다.**

- ◆ 파이썬 내장 모듈인 `time` 모듈을 `import` 키워드를 통해 우리 코드로 임포트합니다.
- ◆ `time` 모듈이 제공하는 `sleep()`을 호출하려면 `time.sleep()`과 같이 함수이름 앞에 모듈이름을 명시해야 합니다. 4라인은 5초간 프로그램을 멈추게 합니다.

우리가 만든 유용한 함수들을 다른 사람들이 사용할 수 있도록 파이썬 모듈로 만들 수 있습니다. 파이썬 모듈을 만드는 방법은 매우 쉽습니다. 다음과 같은 두 개의 함수를 `mylib.py`라는 이름으로 저장합니다.

 mylib.py

```
def add_txt(t1, t2):
    return t1 + ':' + t2

def reverse(x, y, z):
    return z, y, x
```

우리가 새롭게 구현하려는 프로그램에 mylib.py에 있는 두 개의 함수를 이용하려면 다음과 같이 mylib을 임포트하고 함수를 활용하면 됩니다.

```
import mylib

ret1 = mylib.add_txt('대한민국', '1등')
ret2 = mylib.reverse(1, 2, 3.)
print(ret1)          # '대한민국:1등'이 출력됨
print(ret2)          # (3, 2, 1)이 출력됨
```

초급

044



파이썬 패키지 이해하기

- 학습 내용 : 파이썬 모듈의 계층적인 구조인 파이썬 패키지에 대해 이해합니다.
- 힌트 내용 : 파이썬 모듈이 하나의 파이썬 파일이라면 패키지는 디렉터리로 보면 됩니다.

```
1: import mypackage  
2:  
3: ret1 = mypackage.mylib.add_txt('대한민국', '1등')  
4: ret2 = mypackage.mylib.reverse(1, 2, 3)
```

파이썬 모듈을 계층적인 디렉터리 형태로 구성한 것을 **파이썬 패키지**라 합니다. 디렉터리를 파이썬 패키지로 인식하게 하려면 계층적으로 이루어져 있는 각 디렉터리마다 `__init__.py`라는 이름의 파일이 있어야 합니다. `__init__.py`의 내용은 보통 `version = 1.0`과 같이 텍스트 한 줄이면 충분합니다.

실제 패키지를 다음과 같이 직접 만들어 봅니다.

1. `mypackage`라는 이름의 디렉터리를 만듭니다.
2. `mypackage` 디렉터리로 이동합니다.
3. 예제 043에서 구성한 `mylib.py`를 `mypackage` 디렉터리로 복사합니다.
4. `mypackage` 폴더에 `version=1.0` 내용인 `__init__.py` 파일을 생성합니다.
5. 서브 디렉터리를 생성한 후 서브 디렉터리에 하위 패키지를 구성하려면 1~4 과정을 동일하게 반복합니다.

5번까지 마무리되면 `mypackage` 디렉터리는 이제 하나의 패키지로 구성됩니다. `mypackage`를 사용하면 우리의 소스 코드 파일이 있는 디렉터리로 `mypackage` 디렉터리 전체를 복사해옵니다.

- 1 ◆ `mypackage`를 임포트합니다.
- 3~4 ◆ `mypackage`에 있는 `mylib` 모듈의 `add_txt()`를 호출합니다. 마찬가지로 `mypackage`에 있는 `mylib` 모듈의 `reverse()`를 호출합니다.

이 책에서는 특별한 경우를 제외하고 패키지와 모듈을 따로 구분하지 않고 모듈로 부르기로 합니다.

파이썬 모듈 임포트 이해하기 ① (import)

초급

045

- 학습 내용 : 파이썬 모듈을 우리가 작성하는 코드로 임포트하는 방법을 이해합니다.
- 힌트 내용 : 'import 모듈이름'으로 모듈이름에 해당하는 파이썬 모듈을 임포트합니다.



```

1: import time          # 파이썬 내장 모듈인 time을 임포트함
2: import mylib         # 내가 작성한 mylib 모듈을 임포트함
3: import mypackage.mylib # mypackage에 있는 mylib 모듈을 임포트함
4:
5: time.sleep(1)        # time 모듈의 sleep 함수를 이용해 1초간 정지
6: mylib.add_txt('나는', '파이썬이다') # mylib 모듈의 add_txt 함수를 호출
7: mypackage.mylib.reverse(1, 2, 3)    # mypackage.mylib 모듈의 reverse 함수 호출
  
```

이미 만들어져 있는 어떤 함수를 우리가 작성하는 코드에서 자유롭게 활용할 수 있으려면 해당 함수가 포함된 모듈을 임포트해야 합니다. 모듈을 임포트하는 일반적인 방법은 다음과 같습니다.

모듈을 임포트하는 방법

```

import 모듈이름
import 패키지이름.모듈이름
  
```

만약 존재하지 않는 모듈을 임포트하면 다음과 같은 오류가 발생합니다.

```
ImportError: No module named '모듈이름'
```

파이썬에 내장되어 있는 time 모듈을 임포트합니다. time 모듈에 있는 함수를 활용하려면 time.함수 이름과 같이 호출하면 됩니다. ◆ 1

누군가 작성한 함수들을 모아놓은 mylib.py를 임포트합니다. mylib.py에 있는 함수들은 이제 우리의 코드에서 mylib.함수이름으로 호출하여 사용할 수 있습니다. ◆ 2

- 3 ◆ 패키지는 모듈을 계층적으로 모아놓은 것입니다. mypackage에 있는 mylib 모듈을 임포트하는 방법은 3라인처럼 하면 됩니다. 하지만 import mypackage.mylib와 같이 임포트하는 것과 import mypackage로 임포트하는 것은 함수를 호출할 때 크게 다를 것이 없습니다. 예를 들어, add_number()라는 함수가 mylib에 있는 함수라고 하면, 두 가지 방식 모두 add_number()를 호출하는 방법이 같습니다.

```
import mypackage  
# add_number()를 호출하려면 mypackage.mylib.add_number()로 해야 함  
import mypackage.mylib  
# add_number()를 호출하려면 mypackage.mylib.add_number()로 해야 함
```

파이썬 모듈 임포트 이해하기 ② (from~import)

초급

046

- 학습 내용 :** 원하는 모듈만 임포트하여 간단하게 사용할 수 있는 방법에 대해 배웁니다.
- 힌트 내용 :** 계층구조로 되어 있는 복잡한 모듈은 from ~ import로 임포트하면 편리하게 사용할 수 있습니다.



```

1: from time import sleep
2: from mypackage import mylib
3: from mypackage.mylib import reverse
4:
5: sleep(1)                      # time 모듈의 sleep 함수 호출
6: mylib.add_txt('나는', '파이썬이다') # mypackage.mylib 모듈의 add_txt 함수 호출
7: reverse(1, 2, 3)                # mypackage.mylib 모듈의 reverse 함수 호출
  
```

예제 045에서 mypackage.mylib을 임포트하고 mylib에 있는 add_txt()라는 함수를 호출하려면 다음과 같이 해야 합니다.

```

import mypackage
mypackage.mylib.add_txt('나는', '파이썬이다')
  
```

add_txt()를 호출하기 위해 이 함수가 포함된 계층구조를 모두 적어야 하므로 번거로운 일입니다. 우리가 원하는 모듈이나 함수를 계층 경로 모두를 표시하지 않고 간단하게 호출하여 사용할 수 있도록 임포트하는 방법이 있습니다.

```

from 모듈이름 import 함수이름
from 패키지이름 import 모듈이름
  
```

예제를 통해 이해 해봅니다.

- 1 ◆ time 모듈의 sleep() 함수만 임포트합니다. 우리 코드에서 `sleep(1)`과 같이 호출하여 사용할 수 있습니다.
- 2 ◆ mypackage의 mylib 모듈만 임포트합니다. 우리 코드에서 `mylib.add_txt('나는', '파이썬이다')`와 같이 호출하여 사용할 수 있습니다.
- 3 ◆ mypackage.mylib 모듈의 reverse() 함수만 임포트합니다. 우리 코드에서 `reverse(1, 2, 3)`과 같이 호출하여 사용할 수 있습니다.

파이썬 모듈 임포트 이해하기 ③ (import~as)

초급

047

- 학습 내용 : 이름이 긴 모듈에 간단한 별명을 붙이는 방법에 대해 배웁니다.
- 힌트 내용 : 코드에 모듈이름을 자주 사용하는 경우라면 모듈이름에 별명을 붙여 사용하면 편리합니다.



```

1: import mypackage as mp
2: import mypackage.mylib as ml
3:
4: ret1 = mp.mylib.add_txt('대한민국', '1등')
5: ret2 = ml.reverse(1, 2, 3)
  
```

이름이 긴 모듈이나 계층구조가 복잡한 모듈인 경우, 이 모듈에 별명을 붙여 간단하게 호출할 수 있는 방법이 있습니다.

import 이름이 긴 모듈명 as 별명

mypackage를 mp로 축약하여 별명으로 만들었습니다. ◆ 1

mypackage.mylib을 ml로 축약하여 별명으로 만들었습니다. ◆ 2

mp.mylib.add_txt()는 곧 mypackage.mylib.add_txt()와 동일하게 취급됩니다. 마찬가지로 ml.reverse()는 mypackage.mylib.reverse()와 동일하게 취급됩니다. ◆ 4-5

초급

048



파일 열고 닫기(open, close)

- 학습 내용 : 파일을 열고 닫는 방법에 대해 배웁니다.
- 힌트 내용 : 파일은 텍스트 파일과 바이너리 파일이 있으며 파일을 여는 방법에 차이가 있습니다.

```
1: f1 = open('text.txt', 'r')
2: f2 = open('d:/myimages/mypicture1.jpg', 'rb')
3:
4: #-----
5: # 오픈한 파일을 처리하는 코드를 작성함
6: #-----
7:
8: f1.close()
9: f2.close()
```

파일은 텍스트 파일과 바이너리 파일 두 가지 종류가 있습니다. 텍스트 파일은 사람이 읽을 수 있는 글자로 저장된 파일이며 바이너리 파일은 컴퓨터가 읽고 이해할 수 있는 이진 데이터를 기록한 파일입니다. 예를 들어, 윈도우에서 제공하는 메모장 프로그램을 이용하여 내용을 적고 저장하면 텍스트 파일로 저장됩니다. 이미지 뷰어로 볼 수 있는 JPG 이미지 파일은 이미지의 이진 데이터를 JPG 형식의 파일로 저장한 바이너리 파일입니다.

파이썬에서 파일을 다루는 방법은 매우 쉽습니다. 파일을 다루기 위해 가장 먼저 해야 할 일은 파일을 오픈하는 것입니다. 파일을 오픈하기 위해서는 `open()` 함수를 이용하는데, `open()` 함수의 사용 방법은 다음과 같습니다.

`open(파일이름, 모드)`

`open()` 함수의 첫 번째 인자는 파일이름입니다. 파일이름은 파일이 존재하는 절대경로나 상대경로를 입력합니다. 경로 정보 없이 파일이름만 입력하면 프로그램이 구동되는 디렉터리에서 해당 파일을 찾습니다. `open()` 함수의 두 번째 인자는 파일을 오픈하는 모드입니다. 모드는 다음 표와 같습니다.

모드	설명
r 또는 rt	텍스트 모드로 읽기
w 또는 wt	텍스트 모드로 쓰기
a 또는 at	텍스트 모드로 파일 마지막에 추가하기
rb	바이너리 모드로 읽기
wb	바이너리 모드로 쓰기
ab	바이너리 모드로 파일 마지막에 추가하기

`open()` 함수가 성공적으로 파일을 오픈하면 파일을 다룰 수 있는 파일 객체를 리턴합니다. 파일 객체는 파일을 다루는 여러 가지 메소드를 제공합니다. 이에 대한 자세한 내용은 예제 137부터 다루기로 합니다.

프로그램이 구동되는 디렉터리에서 `text.txt` 파일을 찾아 텍스트 읽기 모드로 오픈하고 파일 객체를 `f1`로 둡니다. ◆ 1

파일을 읽기 위해 절대경로를 `open()` 함수의 인자로 입력했습니다. `d:/myimages/mypicture1.jpg`를 바이너리 읽기 모드로 오픈하고 파일 객체를 `f2`로 둡니다. ◆ 2

`open()` 함수를 이용해 파일을 성공적으로 오픈하면 파일 객체를 이용해 파일을 읽거나 쓰는 행위를 수행하는 코드를 작성합니다. ◆ 4-6

파일 다루는 것을 모두 마치면 파일 객체의 `close()`를 이용해 오픈한 파일을 닫습니다. ◆ 8-9

초급

049

클래스 이해하기(class)



- 학습 내용 : 이름을 지정하여 만드는 하나의 독립된 공간인 클래스에 대해 이해합니다.
- 힌트 내용 : 클래스를 활용하면 코드가 간결해지고 체계적으로 구성할 수 있습니다.

소스 : 049.py

```
1: class MyClass:  
2:     var = '안녕하세요'  
3:     def sayHello(self):  
4:         print(self.var)  
5:  
6: obj = MyClass()      # MyClass 인스턴스 객체 생성  
7: print(obj.var)       # '안녕하세요'가 출력됨  
8: obj.sayHello()       # '안녕하세요'가 출력됨
```

객체 지향 프로그래밍에서 중요한 단어가 바로 **클래스**입니다. **클래스**는 프로그래머가 지정한 이름으로 만든 하나의 독립된 공간이며, **이름공간(name space)**이라 부릅니다. 클래스를 구성하는 주요 요소는 클래스에서 변수 역할을 하는 **클래스 멤버**와 함수와 동일한 역할을 하는 **클래스 메소드**입니다. 클래스 멤버나 클래스 메소드는 클래스 공간 내에서 정의되는 것 말고는 보통의 변수나 함수와 그 역할이 비슷합니다.

클래스는 다음과 같이 정의합니다.

class 클래스 이름:
 클래스 멤버 정의
 클래스 메소드 정의

예제는 **MyClass**라는 클래스를 정의하고 이를 활용하는 예시 코드입니다.

◆ 1 MyClass라는 이름의 클래스를 정의합니다.

◆ 2 클래스 멤버는 클래스 메소드 밖에서 정의되는 변수입니다. 클래스 멤버는 클래스 메소드 내에서 정의되는 지역변수나 인스턴스 멤버와는 성질이 다릅니다. 이에 대한 내용은 예제 050에서 다룹니다.

◆ 3~4 클래스 메소드는 클래스 내에서 정의되는 함수입니다. 클래스 메소드는 첫 번째 인자가 반드시 `self`로 시작해야 합니다. `self`는 이 클래스의 인스턴스 객체를 가리키는 참조자입니다.

◆ 6 클래스를 실제 코드에서 활용하려면 인스턴스 객체로 만들어야 합니다. 인스턴스 객체로 만든다는 의미는 선언적인 의미인 클래스를 실제 사용 가능한 형태로 만든다는 뜻입니다. 클래스를 인스턴스 객체로 만드는 방법은 클래스 이름을 함수 호출하듯이 하면 됩니다. `MyClass`를 인스턴스 객체로 만들기 위해서는 `MyClass()`로 호출하면 됩니다. 6라인에서는 `MyClass`의 인스턴스 객체를 `obj`에 지정합니다.

◆ 7 `MyClass`의 인스턴스 객체 `obj`의 멤버 또는 메소드를 호출하는 방법은 다음과 같습니다.

<code>obj.클래스 멤버</code>	# <code>MyClass</code> 의 클래스 멤버
<code>obj.클래스 메소드</code>	# <code>MyClass</code> 의 클래스 메소드

◆ 8 `obj.var`는 `MyClass`의 클래스 멤버 `var`를 나타냅니다. 따라서 `MyClass`의 `var` 값인 ‘안녕하세요’가 출력됩니다.

`obj.sayHello()`는 `MyClass`의 `sayHello()`를 호출합니다. 인스턴스 객체에서 클래스 메소드를 호출할 경우 첫 번째 인자인 `self`는 생략합니다. `MyClass` 선언부 내에서 클래스 멤버를 지시할 때는 `self.클래스 멤버`로 합니다. `MyClass`의 클래스 메소드 `sayHello()`는 클래스 멤버 `var`를 출력하는 함수이므로 `obj.sayHello()`의 결과는 ‘안녕하세요’입니다.

초급

050

클래스 멤버와 인스턴스 멤버 이해하기

- 학습 내용 : 클래스 내에서 선언되는 변수인 클래스 멤버와 인스턴스 멤버에 대해 이해합니다.
- 힌트 내용 : 클래스 멤버는 클래스 메소드 바깥에서 선언되고 인스턴스 멤버는 클래스 메소드 내에서 self와 함께 선언됩니다.

소스 : 050.py

```
1: class MyClass:  
2:     var = '안녕하세요!!'  
3:     def sayHello(self):  
4:         param1 = '안녕'  
5:         self.param2 = '하이'  
6:         print(param1)          # '안녕'이 출력됨  
7:         print(self.var)        # '안녕하세요'가 출력됨  
8:  
9: obj = MyClass()  
10: print(obj.var)           # '안녕하세요'가 출력됨  
11: obj.sayHello()  
12: #obj.param1
```

클래스에서 선언되는 변수는 **클래스 멤버와 인스턴스 멤버**가 있습니다. 클래스 멤버는 클래스 메소드 바깥에서 선언되고 인스턴스 멤버는 클래스 메소드 안에서 self와 함께 선언되는 변수입니다.

예제는 클래스 멤버와 인스턴스 멤버를 선언하고 활용하는 코드 예시입니다.

- 2 ◆ 클래스 멤버는 클래스 내에서 일반 변수를 선언하는 방법과 마찬가지로 변수이름으로 선언하면 됩니다. **MyClass** 내에서 **var**라는 이름의 변수를 선언했습니다. 이 변수는 클래스 메소드 바깥에서 선언되었으므로 클래스 멤버가 됩니다. 클래스 멤버 **var**는 다음과 같이 참조할 수 있습니다.

self.var	# 클래스 메소드 내에서 var를 참조할 경우
MyClass.var	# 클래스 밖에서 클래스 이름만으로 참조할 경우(별로 안 쓰임)
obj.var	# MyClass의 인스턴스 객체 obj에서 var를 참조할 경우

◆ 3-7

클래스 메소드 `sayHello()`는 메소드 내에서만 유효한 지역변수 `param1`을 정의하고 있습니다. `param1`은 `sayHello()` 내에서만 유효한 일종의 지역변수입니다. `sayHello()`는 `param1`의 값과 클래스 멤버인 `var`의 값을 출력합니다. `self.param2`는 `sayHello()` 내에서 선언된 인스턴스 멤버입니다. 이 변수가 초기화되는 시점은 `sayHello()`가 호출되는 시점이므로 `self.param2`를 오류 없이 사용하려면 `sayHello()`가 호출된 이후여야 합니다. `sayHello()`를 호출하기 전에 `self.param2`를 참조하게 되면 다음과 같은 오류가 발생합니다.

```
AttributeError: 'MyClass' object has no attribute 'param2'
```

따라서 인스턴스 멤버의 경우 클래스의 인스턴스 객체가 만들어질 때 자동적으로 호출되는 클래스 생성자에서 선언되는 것이 일반적입니다. 클래스 생성자에 대한 내용은 예제 052에서 다릅니다.

◆ 9-11

`MyClass`의 인스턴스 객체를 만들고 클래스 멤버 `var`를 출력합니다. 10라인에서 `MyClass`의 `sayHello()`를 호출하면 `sayHello()`의 지역변수 `param1`과 `MyClass`의 클래스 멤버 `var`의 값을 화면에 출력합니다.

◆ 12

12라인의 주석을 풀고 실행하면 다음과 같은 오류가 발생합니다.

```
AttributeError: 'MyClass' object has no attribute 'param1'
```

`param1`은 `MyClass`의 클래스 멤버인 `sayHello()`의 지역변수이므로 `MyClass`의 인스턴스 객체인 `obj`의 멤버로 참조가 불가능합니다.

초급

051

클래스 메소드 이해하기



- 학습 내용 : 클래스 내에서 정의되는 함수인 클래스 메소드에 대해 이해합니다.
- 힌트 내용 : 클래스 메소드의 첫 번째 인자는 반드시 self여야 합니다.

소스 : 051.py

```
1: class MyClass:  
2:     def sayHello(self):  
3:         print('안녕하세요')  
4:  
5:     def sayBye(self, name):  
6:         print('%s! 다음에 보자!' %name)  
7:  
8: obj = MyClass()  
9: obj.sayHello()          # '안녕하세요'가 출력됨  
10: obj.sayBye('철수')      # '철수! 다음에 보자!'가 출력됨
```

클래스 내에서 정의되는 함수인 클래스 메소드는 첫 번째 인자가 반드시 self여야 합니다. self는 이 클래스의 인스턴스 객체를 지시하는 참조자입니다. 클래스 메소드의 인자가 필요하면 두 번째 인자부터 정의하면 됩니다.

- 2-3 ◆ MyClass의 클래스 메소드 sayHello()를 정의합니다. 이 메소드는 인자가 없으면 '안녕하세요'라는 문장을 출력합니다.
- 5-6 ◆ MyClass의 클래스 메소드 sayBye(self, name)은 인자가 name입니다.
- 8-9 ◆ MyClass의 인스턴스 객체를 생성하고 obj로 지정합니다. MyClass의 클래스 메소드인 sayHello()와 sayBye()를 호출합니다.

클래스 생성자 이해하기

초급

052

- 학습 내용 : 클래스의 인스턴스 객체가 생성될 때 호출되는 클래스 생성자에 대해 이해합니다.
- 힌트 내용 : 일반적으로 클래스의 인스턴스 멤버를 선언하거나 초기화와 관련된 작업은 클래스 생성자에서 처리합니다.



소스 : 052.py

```

1: class MyClass:
2:     def __init__(self):
3:         self.var = '안녕하세요!'
4:     print('MyClass 인스턴스 객체가 생성되었습니다')
5:
6: obj = MyClass()      # 'MyClass 인스턴스 객체가 생성되었습니다'가 출력됨
7: print(obj.var)       # '안녕하세요'가 출력됨

```

클래스의 인스턴스 객체가 생성될 때 자동적으로 호출되는 메소드가 클래스 생성자입니다. 클래스 생성자는 다음과 같은 이름을 가집니다.

`def __init__(self, *args)`

※ args는 예제 040에서 설명했던 가변 인자입니다.

예제는 클래스의 인스턴스가 생성되면 인사말과 함께 인스턴스 객체가 생성되었다는 메시지를 출력하는 코드입니다.

MyClass의 클래스 생성자를 정의합니다. MyClass의 클래스 생성자는 인자가 없으면 생성자 내에서 `self.var` 인스턴스 멤버를 '안녕하세요!'라는 문자열로 초기화하고 'MyClass 인스턴스 객체가 생성되었습니다'라는 메시지를 출력합니다.

◆ 2-4

MyClass의 생성자에 인자가 없으므로 `MyClass()`와 같이 인스턴스 객체를 생성합니다. 인스턴스 객체가 생성될 때 MyClass의 생성자가 자동으로 호출되므로 `self.var`가 초기화되고 화면에 메시지를 출력합니다.

◆ 6

- 7 ◆ `MyClass`의 생성자에서 초기화한 `self.var`의 값을 출력합니다.

클래스 생성자는 인자를 가질 수 있는데, 생성자에 인자가 있다면 인스턴스 객체를 만들 때 인자에 적당한 값을 대입해 주어야 합니다. `MyClass`의 생성자를 다음과 같이 수정해봅니다.

```
class MyClass:  
    def __init__(self, txt):  
        self.var = txt  
        print('생성자 인자로 전달받은 값은 <' + self.var + '>입니다')
```

`MyClass`의 생성자가 인자를 한 개 가지므로 `MyClass` 인스턴스 객체는 다음과 같이 생성합니다.

```
obj = MyClass('철수')
```

인자를 입력하지 않고 인스턴스 객체를 생성하려고 하면 다음과 같은 오류가 발생합니다.

```
TypeError: __init__() missing 1 required positional argument: 'txt'
```

클래스 소멸자 이해하기

초급

053

- 학습 내용 :** 클래스의 인스턴스 객체가 메모리에서 제거될 때 자동으로 호출되는 클래스 소멸자에 대해 이해합니다.
- 힌트 내용 :** 클래스 소멸자는 인스턴스 객체에서 사용한 자원을 해제하기 위해 자동으로 호출됩니다.



소스 : 053.py

```

1: class MyClass:
2:     def __del__(self):
3:         print('MyClass 인스턴스 객체가 메모리에서 제거됩니다')
4:
5: obj = MyClass()
6: del obj      # 'MyClass 인스턴스 객체가 메모리에서 제거됩니다'가 출력됨

```

클래스 인스턴스 객체가 메모리에서 제거될 때 자동적으로 호출되는 클래스 메소드가 클래스 소멸자입니다. 클래스 소멸자는 다음과 같은 이름을 가집니다.

`__del__(self):`

인스턴스 객체를 메모리에서 제거하려면 `del` 키워드를 이용합니다.

인스턴스 객체 제거

`del <인스턴스 객체>`

예제는 클래스 인스턴스를 메모리에서 제거할 때 클래스 소멸자에서 메시지를 출력하는 코드입니다.

클래스 소멸자에서 `print()`를 호출하여 메시지를 출력하도록 합니다.

◆ 2-3

생성된 `MyClass` 인스턴스 객체 `obj`를 `del` 키워드로 메모리에서 제거합니다. 이때 클래스 소멸자가 자동으로 호출되므로 3라인에서 작성된 메시지가 화면에 출력됩니다.

◆ 6

초급

054

클래스 상속 이해하기



- 학습 내용 : 부모클래스의 모든 기능들을 자식클래스로 상속하는 개념에 대해 이해합니다.
- 힌트 내용 : 상속해주는 클래스를 부모클래스 또는 슈퍼클래스라 하고, 상속받는 클래스를 자식클래스 또는 서브클래스라 부릅니다.

소스 : 054.py

```
1: class Add:  
2:     def add(self, n1, n2):  
3:         return n1+n2  
4:  
5: class Calculator(Add):  
6:     def sub(self, n1, n2):  
7:         return n1-n2  
8:  
9: obj = Calculator()  
10: print(obj.add(1, 2))      # 3이 출력됨  
11: print(obj.sub(1, 2))      # -1이 출력됨
```

클래스는 상속이라는 특성을 가지는 이름공간입니다. 클래스에서 상속이란 어떤 클래스가 가지고 있는 모든 멤버나 메소드를 상속받는 클래스가 모두 사용할 수 있도록 해주는 것입니다. 상속을 해주는 클래스를 **부모클래스** 또는 **슈퍼클래스**라 부르고 상속을 받는 클래스를 **자식클래스** 또는 **서브클래스**라 부릅니다. 부모클래스로부터 상속을 받아 자식클래스를 정의하는 방법은 다음과 같습니다.

class 자식클래스(부모클래스):

자식클래스는 부모클래스에서 정의된 모든 멤버나 메소드들을 그대로 상속받습니다. 만약 자식클래스에서 부모클래스로부터 상속받은 멤버나 메소드와 동일한 이름의 멤버나 메소드가 존재하면 자식클래스에서 정의된 것이 우선합니다.

예제는 두 수를 더하는 클래스를 상속받은 자식클래스를 정의하고 자식클래스에서 두 수를 빼는 클래스 메소드를 정의하는 코드입니다.

Add라는 이름의 클래스를 정의합니다. 이 클래스는 두 수를 더한 값을 리턴하는 add()라는 클래스 메소드가 있습니다.

◆ 1-3

Calculator라는 이름의 클래스는 Add 클래스를 상속받아 정의됩니다. 따라서 Calculator 클래스는 Add 클래스의 add() 메소드를 그대로 물려받습니다. Calculator 클래스에서 두 수를 뺀 값을 리턴하는 sub()라는 클래스 메소드를 정의합니다.

◆ 5-7

Calculator 클래스의 인스턴스 객체를 생성하고 이 객체의 add(1, 2)를 호출합니다. add()는 Calculator의 부모클래스인 Add 클래스로부터 상속받은 것입니다. 그리고 Calculator에서 정의한 sub() 메소드를 호출합니다.

◆ 9-11

자식클래스는 여러 개의 부모클래스로부터 상속받을 수 있습니다. 이를 **다중상속**이라 합니다. 다중 상속 클래스를 정의하는 방법은 다음과 같습니다.

```
class 자식클래스(부모클래스1, 부모클래스2, ..)
```

예제 코드에 다음과 같이 Multiply 클래스를 추가하고 Calculator 클래스를 Add 클래스와 Multiply 클래스로부터 다중상속하도록 수정해봅니다.

소스 : 054-1.py

```
class Add:
    def add(self, n1, n2):
        return n1+n2

class Multiply:
    def multiply(self, n1, n2):
        return n1*n2

class Calculator(Add, Multiply):
    def sub(self, n1, n2):
        return n1-n2

obj = Calculator()
print(obj.add(1, 2))          # 30 | 출력됨
print(obj.multiply(3, 2))      # 60 | 출력됨
```

초급

055

예외처리 이해하기 ① (try~except)



• 학습 내용 : 코드에서 예외 상황이 발생했을 때 프로그램을 종료시키지 않고 예외를 처리하는 방법에 대해 배웁니다.

• 힌트 내용 : 예외 발생 가능성 있는 코드는 try와 except 사이에 위치시킵니다.

소스 : 055.py

```
1: try:  
2:     print('안녕하세요.')  
3:     print(param)  
4: except:  
5:     print('예외가 발생했습니다!')
```

프로그램을 작성하다보면 뜻하지 않은 오류가 발생하는 코드가 있을 수 있습니다. 프로그램이 실행되는 동안 오류가 발생하면 프로그램이 더 이상 진행될 수 없는 상태가 되는데 이를 예외(Exception) 상황이라고 합니다.

프로그램에 예외가 발생하더라도 프로그램을 중단시키지 않고 예외에 대한 적절한 처리를 하여 프로그램을 계속 진행시킬 수 있도록 하는 구문이 try~except입니다.

프로그램의 논리적 오류가 발생할 가능성이 큰 부분을 try~except 사이에 두고 예외가 발생할 경우, except 부분에서 적절한 처리를 해줍니다.

예제는 정의되지 않은 변수를 호출하여 발생하는 예외 상황에 대한 처리 예시입니다.

- 3 ◆ 정의되지 않은 변수 param을 화면에 출력하는 코드는 다음과 같은 예외를 발생시키고 프로그램을 중단합니다.

NameError: name 'param' is not defined

하지만 3라인은 try~except 내부에 있으므로 위와 같은 예외가 발생하면 except 부분으로 넘어갑니다. 실제 활용에서는 except에서 예외 발생 내용이나 원인을 파악하기 위해 적절한 처리를 해줍니다. 이에 대한 내용은 예제 057에서 다루도록 합니다. 예제를 실행하면 다음과 같은 결과가 표시됩니다.



초급

056

예외처리 이해하기 ② (try~except~else)

- 학습 내용 : 코드에서 예외가 발생하지 않았을 때만 실행해야 하는 코드 처리 방법에 대해 배웁니다.
- 힌트 내용 : try~except~else에서 else 다음에 예외가 발생하지 않았을 때만 실행하는 코드를 위치 시킵니다.

소스 : 056.py

```
1: try:  
2:     print('안녕하세요.')  
3:     print(param)  
4: except:  
5:     print('예외가 발생했습니다!')  
6: else:  
7:     print('예외가 발생하지 않았습니다.')
```

어떤 로직을 수행했을 때 오류 상황이 아닐 경우에만 어떤 작업을 수행하는 코드를 작성해야 할 때가 있습니다. 이럴 경우 try~except~else 구문을 활용합니다.

3 ◆ 예제에서 param은 정의되지 않은 변수이므로 이 부분에서 예외가 발생합니다. 따라서 except 부분의 코드가 실행되고 화면에는 '예외가 발생했습니다!'라는 메시지가 출력됩니다. 예외가 발생했으므로 else 부분은 실행되지 않습니다. 3라인을 주석 처리하고 다시 실행하면 예외 부분이 없어졌기 때문에 except 부분은 실행되지 않고 else 부분이 실행되어 화면에 '예외가 발생하지 않았습니다.'가 출력됩니다.

결과

안녕하세요.
예외가 발생했습니다!

3라인을 주석 처리하였을 때는 다음과 같은 결과가 표시됩니다.

결과

안녕하세요.
예외가 발생하지 않았습니다.

예외처리 이해하기 ③

(try~except~finally)

초급

057

- 학습 내용 : 코드에서 예외 발생 유무와 상관없이 무조건 실행해야 하는 코드 처리 방법에 대해 배웁니다.
- 힌트 내용 : try~except~finally에서 finally 다음에 무조건 실행해야 하는 코드를 위치시킵니다.



소스 : 057.py

```

1: try:
2:     print('안녕하세요.')
3:     print(param)
4: except:
5:     print('예외가 발생했습니다!')
6: finally:
7:     print('무조건 실행하는 코드')

```

오류 발생 유무와 상관없이 어떤 코드를 무조건 실행시키려면 try~except~finally 구문을 활용합니다. 무조건 실행시키려는 코드는 finally 부분에 작성하면 됩니다.

3라인에서 정의되지 않은 변수인 **param**을 화면에 출력하려고 했으므로 이 부분에서 예외가 발생되어 except 부분이 실행됩니다. 그리고 finally는 예외가 발생하든 발생하지 않든 무조건 실행되는 부분입니다. 따라서 화면에는 다음과 같은 결과가 표시됩니다.



결과

안녕하세요.

예외가 발생했습니다!

무조건 실행하는 코드

3라인을 주석 처리하고 코드를 실행해도 무조건 실행하는 코드가 화면에 출력됩니다.

초급

058

예외처리 이해하기 ④ (try~except Exception as e)

- 학습 내용 : 코드에서 예외 발생 내용을 확인하고자 하는 경우 코드 처리 방법에 대해 배웁니다.
- 힌트 내용 : try~except Exception as e에서 e가 예외 발생 내용이 담긴 변수입니다.

소스 : 058.py

```
1: try:  
2:     print(param)  
3: except Exception as e:  
4:     print(e)      # name 'param' is not defined가 출력됨
```

코드에서 예외가 발생하면 이에 대한 자세한 내용을 파악하는 것이 중요합니다. 파일은 발생 가능한 예외에 대해 exception 객체로 미리 정의해두고 있는데 이에 대한 내용은 다음 링크에서 확인할 수 있습니다.

<https://docs.python.org/3/library/exceptions.html#bltin-exceptions>

- 2-3 ◆ 예제는 param이라는 정의되지 않은 변수를 코드에서 쓰고 있습니다. 이는 파일에서 NameError 예외를 발생시키고 except Exception as e는 NameError 객체를 e라는 이름으로 접근할 수 있게 해줍니다.
- 4 ◆ NameError 객체를 출력해보면 다음과 같은 메시지가 표시됩니다.



name 'param' is not defined

예외처리 이해하기 5 (try~except 특정 예외)

초급

059

- 학습 내용 : 코드에서 특정 예외가 발생했을 때만 코드를 처리하는 방법에 대해 배웁니다.
- 힌트 내용 : 특정 예외는 058에서 언급한 exception 객체를 말합니다.



소스 : 059.py

```

1: import time
2: count = 1
3: try:
4:     while True:
5:         print(count)
6:         count += 1
7:         time.sleep(0.5)
8: except KeyboardInterrupt:      # Ctrl+C가 입력되면 발생하는 오류
9:     print('사용자에 의해 프로그램이 중단되었습니다.')

```

코드에서 특정 예외 상황이 발생했을 때만 except에서 처리할 수 있는 방법이 있는데, except 다음에 특정 예외를 명시해주면 됩니다.

예제 코드는 사용자가 키보드로 **Ctrl+C**를 누를 때까지 1씩 증가하는 숫자를 0.5초마다 화면에 표시합니다. **Ctrl+C**를 누를 때 발생하는 예외는 `KeyboardInterrupt`입니다.

`sleep()` 함수를 이용하기 위해 `time` 모듈을 임포트합니다. ◆ 1

`count`의 초기값을 1로 설정합니다. ◆ 2

`while True`는 무한 반복문을 의미합니다. `count` 값을 출력하고 `count`를 1 증가합니다. 0.5초 동안 멈추고 다시 `count` 값을 출력하고 1 증가시키는 로직을 무한 반복합니다. 사용자에 의해 **Ctrl+C**가 입력되면 `try` 내부에 있는 로직에서 `KeyboardInterrupt` 예외가 발생하게 되고 `except KeyboardInterrupt` 부분으로 제어가 넘어갑니다. 이 부분은 별다른 처리없이 화면에 '사용자에 의해 프로그램이 중단되었습니다.'를 출력하고 프로그램을 종료합니다. ◆ 3-7

3

P A R T

중급

Python 프로그래밍 실력 다지기

중급

060

사용자 입력받기(input)

- **학습 내용 :** 사용자가 입력하는 값을 프로그램에서 받는 방법을 배웁니다.
- **힌트 내용 :** `input()`은 사용자가 [Enter]를 누를 때까지의 값을 입력받고 그 값을 문자열로 리턴합니다.

소스 : 060.py

```
1: k = input('<값>을 입력하세요: ')
2: print('당신이 입력한 값은 <' + k + '>입니다.')
```

- ◆ 파이썬 내장 함수 `input()`은 사용자가 키보드로 입력한 값을 문자열로 리턴합니다. `input()`의 인자는 사용자 입력을 돋기 위한 안내 문구나 힌트 등을 표시하는 문자열이 됩니다. 1라인은 화면에 '<값>을 입력하세요:'를 출력하고 사용자 입력을 기다립니다. 사용자가 키보드로 값을 입력하고 [Enter]를 누르면 `input()`은 사용자가 입력한 값을 문자열로 리턴하며, 변수 `k`에 대입합니다.
- ◆ 사용자가 '안녕하세요 파이썬'이라는 문장을 입력하면 다음과 같은 결과가 표시됩니다.



당신이 입력한 값은 <안녕하세요 파이썬>입니다.

자료형 확인하기(type)

중급

061



- 학습 내용 : 자료형을 확인하는 방법에 대해 배웁니다.
- 힌트 내용 : 파이썬 내장 함수 type()은 인자로 입력된 자료형을 리턴합니다.

소스 : 061.py

```

1: numdata = 57
2: strdata = '파이썬'
3: listdata = [1, 2, 3]
4: dictdata = {'a':1, 'b':2}
5:
6: def func():
7:     print('안녕하세요.')
8:
9: print(type(numdata))
10: print(type(strdata))
11: print(type(listdata))
12: print(type(dictdata))
13: print(type(func))

```

파이썬의 자료형은 하나의 클래스입니다. 파이썬은 숫자나 문자, 문자열, 리스트, 튜플, 사전, 함수 등을 각각 하나의 클래스로 취급됩니다. 코드를 작성하다가 변수이름만 보고 이 자료가 어떤 자료형인지 확인해야 할 경우가 있습니다. 이때 파이썬 내장함수인 type()을 활용하면 자료형을 쉽게 확인할 수 있습니다.

예제는 정수, 문자열, 리스트, 사전, 함수에 대해 자료형을 확인하는 코드입니다. 이 코드를 실행하면 다음과 같은 결과가 나옵니다.



```
<class 'int'>
<class 'str'>
<class 'list'>
<class 'dict'>
<class 'function'>
```

나눗셈에서 나머지만 구하기(%)

중급

062

- 학습 내용 : 나누기 연산에서 나머지만 구하는 방법을 배웁니다.
- 힌트 내용 : 나머지만 구하는 연산자는 %입니다.



소스 : 062.py

```

1: a = 11113
2: b = 23
3: ret = a%b
4: print('<%d>를 <%d>로 나누면 <%d>가 나머지로 남습니다.' %(a, b, ret))

```

나누기 연산에서 나머지만 구하는 연산을 모듈로(modulo) 연산이라 하며 기호는 mod로 표시합니다. 예를 들어 12는 3으로 나누어 떨어지므로 나머지가 0이 되며, 12를 5로 나누면 나머지는 2가 됩니다. 이를 수학식으로 표현하면 다음과 같습니다.

12 mod 3 = 0
12 mod 5 = 2

파이썬에서 모듈로 연산 기호는 % 연산자입니다. 위 수학식은 파이썬 코드에서 다음과 같이 표현됩니다.

12 % 3 = 0
12 % 5 = 2

예제는 11113을 23으로 나누었을 때 나머지를 구하는 코드입니다. 이 코드의 실행 결과는 다음과 같습니다.

◆ 1-4



<11113>를 <23>로 나누면 <4>가 나머지로 남습니다.

중급

063

몫과 나머지 구하기(divmod)



- 학습 내용 : 두 정수의 나누기 연산에서 몫과 나머지를 구하는 방법을 배웁니다.
- 힌트 내용 : 파이썬 내장함수 divmod()는 두 정수의 나눗셈에서 몫과 나머지를 리턴하는 함수입니다.

소스 : 063.py

```
1: a = 11113
2: b = 23
3: ret1, ret2 = divmod(a, b)
4: print('<%d / %d>는 몫이 <%d>, 나머지가 <%d>입니다.' %(a, b, ret1, ret2))
```

예제는 11113을 23으로 나누었을 때 몫과 나머지를 구하는 코드입니다.

- 3-4 ◆ 파이썬 내장함수 divmod()는 두 개의 정수를 인자로 받습니다. 첫 번째 인자를 두 번째 인자로 나눈 결과를 (몫, 나머지)와 같이 튜플로 리턴합니다. 코드 실행 결과는 다음과 같습니다.



<11113/23>는 몫이 <483>, 나머지가 <4>입니다.

10진수를 16진수로 변환하기(hex)

중급

064

- 학습 내용 : 10진수 정수를 16진수 정수로 변환하는 방법에 대해 배웁니다.
- 힌트 내용 : 파이썬 내장함수 hex()는 10진수를 16진수로 변환한 값을 문자열로 리턴합니다.



소스 : 064.py

```

1: h1 = hex(97)      # h1은 문자열 '0x61'
2: h2 = hex(98)      # h2는 문자열 '0x62'
3: ret1 = h1 + h2
4: print(ret1)       # '0x610x62' 가 출력됨
5: a = int(h1, 16)
6: b = int(h2, 16)
7: ret2 = a + b      # ret2는 10진수 195가 됨
8: print(hex(ret2))  # '0xc3' 가 출력됨

```

파이썬 내장함수 hex()는 인자로 입력된 10진수 정수를 16진수로 변환해서 문자열로 리턴합니다. hex()로 변환한 16진수를 덧셈이나 뺄셈을 수행하고자 하면 파이썬 내장함수 int()를 이용해 숫자로 변환해야 합니다.

파이썬 내장함수 hex()는 10진수 97과 98을 16진수로 변환하고 이를 문자열로 리턴하므로 h1과 h2는 각각 '0x61'과 '0x62'가 됩니다. 이는 숫자가 아니라 문자열입니다.

◆ 1-2

h1과 h2를 더하면 '0x610x62'가 결과로 나옵니다.

◆ 3-4

hex()로 변환한 값은 int()를 이용해 숫자로 변환할 수 있습니다. '0x61'과 '0x62'는 16진수이므로 int('0x61', 16)과 같이 int()의 두 번째 인자에 정수로 변환할 수가 16진수임을 지정합니다. 두 번째 인자가 없으면 정수로 변환할 수를 10진수로 처리합니다. '0x61'은 10진수 형식이 아니므로 int('0x61')를 호출하면 다음과 같은 오류가 발생합니다.

◆ 5-6

```
ValueError: invalid literal for int() with base 10: '0x61'
```

`int()`는 `h1, h2`를 10진수 값으로 변환하고 `a, b`에 대입합니다. `int()`는 예제 066에서 다릅니다.

- 7-8 ◆ `a`와 `b`를 더하면 10진수로 195가 됩니다. 195를 `hex()`로 16진수로 변환하면 문자열 '`0xc3`'이 됩니다.



N O T E | 문자열 포맷팅을 이용해 10진수를 16진수로 출력하기 |

문자열 포맷팅에 사용되는 `%X`와 `%x`는 16진수를 나타내는 포맷 문자열입니다. `%X`는 16진수를 대문자로 표현하며, `%x`는 16진수를 소문자로 표현합니다.

```
n = 159  
print('%X' %n)  
print('%x' %n)
```

위 코드를 실행하면 다음과 같은 결과가 출력됩니다.

결과

```
9F  
9f
```

포맷 문자열을 `%02X`, `%02x`와 같이 사용하면 한 자리수 16진수인 경우 앞에 0을 붙여 표현합니다.

```
n = 11  
print('%02X' %n)  
print('%02x' %n)
```

위 코드를 실행하면 다음과 같은 결과가 출력됩니다.

결과

```
0B  
0b
```

10진수를 2진수로 변환하기 (bin)

중급

065

- 학습 내용 : 10진수 정수를 2진수로 변환하는 방법에 대해 배웁니다.
- 힌트 내용 : 파이썬 내장함수 bin()은 10진수를 2진수로 변환한 값을 문자열로 리턴합니다.



소스 : 065.py

```

1: b1 = bin(97)      # b1은 문자열 '0b11000001'
2: b2 = bin(98)      # b2는 문자열 '0b11000010'
3: ret1 = b1 + b2
4: print(ret1)       # '0b110000010b11000010'가 출력됨
5: a = int(b1, 2)
6: b = int(b2, 2)
7: ret2 = a + b
8: print(bin(ret2))  # '0b11000011'이 출력됨

```

파이썬 내장함수 bin()은 파이썬 3에서 새롭게 추가된 함수입니다. bin()은 인자로 입력된 숫자를 2진수로 변환하여 그 값을 문자열로 리턴합니다. 파이썬에서 2진수는 숫자 앞에 0b를 붙여서 나타냅니다. hex() 함수와 마찬가지로 bin()의 결과도 문자열이기 때문에 bin()의 결과를 가지고 수학 연산을 하려면 int() 함수를 이용해 10진수로 변환한 다음 계산을 하고 bin()으로 다시 2진수로 변환합니다.

파이썬 내장함수 bin()은 10진수 97과 98을 2진수로 변환하고 이를 문자열로 리턴하므로 b1과 b2는 각각 문자열 '0b11000001'과 '0b11000010'가 됩니다.

◆ 1-2

b1과 b2를 더하면 '0b110000010b11000010'이 됩니다.

◆ 3-4

bin()으로 변환한 값은 int()를 이용해 숫자로 변환할 수 있습니다. 원리는 예제 064에서 설명한 것과 비슷합니다.

◆ 5-6

중급

066

2진수, 16진수를 10진수로 변환하기(int)

- 학습 내용 : 2진수나 16진수 정수를 10진수로 변환하는 방법을 배웁니다.
- 힌트 내용 : 파이썬 내장함수 int()는 인자로 입력된 숫자나 문자열을 10진수로 바꾸어 줍니다.

소스 : 066.py

```
1: bnum = 0b11110000; bstr = '0b11110000'  
2: onum = 0o360; ostr = '0o360'  
3: hnum = 0xf0; hstr = '0xf0'  
4: b1 = int(bnum); b2 = int(bstr, 2)      # b2 = int(bstr, 0)로도 가능  
5: o1 = int(onum); o2 = int(ostr, 8)      # o2 = int(ostr, 0)로도 가능  
6: h1 = int(hnum); h2 = int(hstr, 16)      # h2 = int(hstr, 0)로도 가능  
7: print(b1); print(b2)  
8: print(o1); print(o2)  
9: print(h1); print(h2)
```

파이썬 내장함수 int()는 2진수, 8진수, 16진수 정수를 10진수 정수로 변환합니다. 파이썬 내장함수 bin()이나 hex()로 변환된 2진수나 16진수 문자열도 int()를 이용하면 10진수로 변환할 수 있는데 int()의 두 번째 인자에 2진수, 16진수임을 지정해주면 됩니다.

- 1-3 ◆ 예제는 2진수로 된 숫자와 문자열, 8진수로 된 숫자와 문자열 그리고 16진수로 된 숫자와 문자열을 선언하고 int()를 이용해 10진수로 변환하는 코드입니다.
- 4-6 ◆ 0b, 0o, 0x가 붙은 숫자는 int()가 2진수, 8진수, 16진수임을 자동으로 인식하고 10진수로 변환하기 때문에 두 번째 인자를 지정해주지 않아도 됩니다. 문자열로 된 2진수, 8진수, 16진수는 int()의 두 번째 인자에 2, 8, 16을 지정해야 제대로 변환합니다. int()의 두 번째 인자로 0을 지정하면 첫 번째 인자의 문자열 그대로 숫자로 변환하고 10진수로 변환한 결과를 리턴합니다. 예를 들어, int('0xf0', 0)은 '0xf0'를 0xf0로 인식하고 이를 10진수로 바꾸어 줍니다. 따라서 int('0xf0', 16)과 int('0xf0', 0)은 동일한 결과를 리턴합니다.
- 7-9 ◆ 모두 240이 결과로 나옵니다.

절대값 구하기(abs)

중급

067

- 학습 내용 : 주어진 수의 절대값을 구하는 방법을 배웁니다.
- 힌트 내용 : 파이썬 내장함수 abs()는 인자로 입력된 값의 절대값을 구해줍니다.



소스 : 067.py

```

1: abs1 = abs(-3)      # 정수의 절대값
2: abs2 = abs(-5.72)    # 실수의 절대값
3: abs3 = abs(3+4j)     # 복소수의 절대값
4: print(abs1)          # 3이 출력됨
5: print(abs2)          # 5.72가 출력됨
6: print(abs3)          # 5.0이 출력됨
  
```

파이썬 내장함수 abs()는 인자로 입력된 값의 절대값을 리턴합니다. 만약 복소수가 인자로 입력되면 복소수의 크기를 리턴합니다. 복소수 $a+bi$ 의 크기는 $\sqrt{a^2+b^2}$ 입니다.

예제는 정수 -3, 실수 -5.72, 복소수 3+4j의 절대값을 구합니다. 각각의 절대값들은 3, 5.72, 5.0입니다.

◆ 1~6

중급

068

반올림수 구하기(round)

- 학습 내용 : 수치형 자료의 반올림에 대해 이해합니다.
- 힌트 내용 : 파이썬 내장함수 round()는 특정 자리까지 반올림한 수를 계산해줍니다.

소스 : 068.py

```
1: ret1 = round(1118)          # 소수점 첫째자리에서 반올림해줌
2: ret2 = round(16.554)         # 소수점 첫째자리에서 반올림해줌
3: ret3 = round(1118, -1)       # 1자리에서 반올림해줌
4: ret4 = round(16.554, 2)       # 소수점 셋째자리에서 반올림해줌
5: print(ret1)    # 1118
6: print(ret2)    # 17
7: print(ret3)    # 1120
8: print(ret4)    # 16.55
```

파이썬 내장함수 round()는 인자로 입력된 수치형 자료를 지정된 자리수에서 반올림한 결과를 리턴합니다.

- ◆ 디폴트로 round()는 입력된 숫자의 소수점 첫째자리에서 반올림한 수를 리턴합니다. 1118은 소수점이 없으므로 ret1의 값은 1118 그대로입니다.
- ◆ 16.554를 소수점 첫째자리에서 반올림하면 17이 됩니다. ret2의 값은 17입니다.
- ◆ round()의 두 번째 인자는 반올림을 실시할 자릿수를 나타내는데, 디폴트 값은 0입니다. 두 번째 인자가 0일 때 소수점 첫째자리에서 반올림을 실시하므로 1, 2, 3일 때는 각각 소수점 둘째자리, 소수점 셋째자리, 소수점 넷째자리에서 반올림을 실시합니다. 만약 두 번째 인자의 값이 -1, -2이면 1자리, 100자리에서 반올림을 실시합니다. round(1118, -1)은 1118의 1자리인 8에서 반올림을 실시하므로 ret3의 값은 1120이 됩니다.
- ◆ round(16.554, 2)는 소수점 셋째자리에서 반올림하므로 ret4의 값은 16.55가 됩니다.

실수형 자료를 정수형 자료로 변환하기(int)

중급

069

- 학습 내용 : 실수형 자료를 정수형 자료로 변환하는 방법에 대해 배웁니다.
- 힌트 내용 : 파이썬 내장함수 int()는 실수형 자료가 입력되면 정수부분에 해당하는 숫자를 리턴합니다.

 소스 : 069.py

```

1: idata1 = int(-5.4)
2: idata2 = int(1.78e1)
3: idata3 = int(171.56)
4: print(idata1)      # -5가 출력됨
5: print(idata2)      # 170이 출력됨
6: print(idata3)      # 171이 출력됨

```

코드 작성 시 수학 연산을 하다보면 정수끼리만 계산해야 하는 경우가 있습니다. 이때 우리가 가진 데이터가 실수형이라고 하면 실수형 자료를 정수형으로 변환한 후에 계산해 주어야 합니다. 파이썬 내장함수 int()는 인자로 입력된 실수형 자료를 정수형 자료로 변환해 줍니다. int()는 입력된 실수형 자료의 소수부분은 버리고 정수부분만 취하여 정수값으로 리턴합니다.

-5.4를 int()로 정수형 자료로 변환하면 -5가 됩니다.

◆ 1

1.78e1은 17.8이므로 int()로 정수형 자료로 변환하면 17이 됩니다.

◆ 2

171.56을 int()로 정수형 자료로 변환하면 171이 됩니다.

◆ 3

중급

070

정수형 자료를 실수형 자료로 변환하기(float)

- **학습 내용 :** 정수형 자료를 실수형 자료로 변환하는 방법에 대해 배웁니다.
- **힌트 내용 :** 파이썬 내장함수 float()는 정수형 자료가 입력되면 정수에 .0을 붙여 실수형 자료로 리턴합니다.

소스 : 070.py

```
1: fdata = float(10)
2: print(fdata)      # 10.0으로 출력됨
```

이미지 처리나 공학용 프로그램을 작성할 때 실수형 자료끼리만 계산해야 하는 경우가 많습니다. 이때 우리가 가진 데이터가 정수형이라고 하면 정수형 자료를 실수형으로 변환한 후에 계산해 주어야 합니다. 파이썬 내장함수 float()는 인자로 입력된 정수형 자료를 실수형 자료로 변환해 줍니다.

- 1-2 ◆ 정수형 자료 10을 float()을 이용해 실수형으로 변환합니다. 이 값을 출력해보면 10.0으로 표시되어 실수형 자료로 변환되었음을 알 수 있습니다.