2장 문자열 처리 프로그램

2.1 회문 여부 판별

프로그램 개요

- 문자열을 입력 받아 회문인지 여부를 판별하는 프로그램 작성

  - 입력 : 문자열 s
  - 출력 : s의 회문 여부

- 회문(回文, palindrome)
  - 앞에서부터 읽는 것과 뒤에서부터 읽는 것이 동일한 단어나 문장
  - 회문인지 여부를 판별할 때에는 보통 띄어쓰기(space)나 쉼표(comma), 느낌표, 물음표, 따옴표와 같은 문장 부호를 무시하고, 영어에서는 대문자와 소문자를 구분하지 않음
  - 한국어 회문 : "기러기", "다시 합창합시다", "여보 안경 안 보여", "사전 사! 영한사전 사! 영영사전 사! 한영사전 사!“
  - 영어 회문 : “madam", "reviver", "Was it a car or a cat I saw", "Able was I ere I saw Elba", "A man, a plan, a canal – Panama!"

파이썬 문법

문자열 다루기 (1)

- 문자(character)는 'a', 'b', 'c'와 같이 글자 하나하나를 의미하고, 문자열(string)은 문자들이 여러 개 모여 있는 것을 의미
- 변수에 문자열을 저장하는 예

```
>>> s = 'Python'
>>> s
'Python'
>>> type(s)
<class 'str'>
```

- 문자열 'Python'의 인덱스

P	y	t	h	o	n
[0]	[1]	[2]	[3]	[4]	[5]
[-6]	[-5]	[-4]	[-3]	[-2]	[-1]

문자열 다루기 (2)

- 문자열 인덱싱의 예

```
>>> s = 'Python'
>>> s[0]
'P'
>>> s[5]
'n'
>>> s[6]
Traceback (most recent call last):
File "<pyshell#75>", line 1, in <module>
s[6]
IndexError: string index out of range
>>> s[-1]
'n'
>>> s[-6]
'P'
>>> s[-7]
Traceback (most recent call last):
File "<pyshell#78>", line 1, in <module>
s[-7]
IndexError: string index out of range
```

문자열 다루기 (3)

- 문자열 슬라이싱

```
문자열[시작범위:끝범위:증가치]
```

- 문자열 슬라이싱의 예

```
>>> s[:]
'Python'
>>> s[0:2]
'Py'
>>> s[2:]
'thon'
>>> s[:3]
'Pyt'
>>> s[-2:]
'on'
>>> s[-5:-2]
'yth'
>>> s[::2]
'Pto'
```

문자열 다루기 (4)

- 문자열 접합(string concatenation)

```
>>> s1 = 'Python '
>>> s2 = 'Programming'
>>> s3 = s1 + s2
>>> s3
'Python Programming'
```

- 문자열 접합 프로그램

```
l = ['a', 'b', 'c', 'd', 'e', 'f']
n = len(l)
s = ''
for i in range(n):
    s += l[i]
print(s)
```

문자열 다루기 (5)

- 문자열의 내장 메소드

메소드	동작
lower() upper() swapcase() capitalize() title()	대문자를 소문자로 변환 소문자를 대문자로 변환 소문자는 대문자로, 대문자는 소문자로 변환 첫 문자를 대문자로 변환 단어의 첫 문자를 대문자로 변환
islower() isupper()	모든 문자가 소문자이면 True 반환 모든 문자가 대문자이면 True 반환
count(s)	s가 몇 번 나타났는지 개수 반환
find(s) index(s)	s가 나타난 첫 번째 위치 반환 (없다면 -1 반환) s가 나타난 첫 번째 위치 반환 (없다면 예외 발생)
isalnum() isalpha() isnumeric()	모든 문자가 알파벳 혹은 숫자이면 True 반환 모든 문자가 알파벳이면 True 반환 모든 문자가 숫자이면 True 반환
split(s)	s를 기준으로 문자열을 분리

문자열 다루기 (6)

- 문자열의 내장 메소드 사용 예 (1)

```
>>> s = 'Able was I ere I saw Elba'
>>> s1 = s.upper()
>>> s1
'ABLE WAS I ERE I SAW ELBA'
>>> s1.isupper()
True
>>> s1.islower()
False
>>> s.swapcase()
'aBLE WAS i ERE i SAW eLBA'
>>> s2 = s.lower()
>>> s2
'able was i ere i saw elba'
>>> s2.capitalize()
'Able was i ere i saw elba'
>>> s2.title()
'Able Was I Ere I Saw Elba'
```

문자열 다루기 (7)

- 문자열의 내장 메소드 사용 예 (2)

```
>>> s.count('I')
2
>>> s.find('was')
5
>>> s.find('me')
-1
>>> s.index('I')
9
>>> s.index('me')
Traceback (most recent call last):
File "<pyshell#36>", line 1, in <module>
s.index('me')
ValueError: substring not found
```

문자열 다루기 (8)

- 문자열의 내장 메소드 사용 예 (3)

```
>>> s2 ='123ABCabc'
>>> s2.isalnum()
True
>>> s2.isalpha()
False
>>> s2.isnumeric()
False
>>> s3 = 'ABCabc'
>>> s3.isalpha()
True
>>> s4 = '123'
>>> s4.isnumeric()
True
```

문자열 다루기 (9)

- 문자열의 내장 메소드 사용 예 (4)

```
>>> s = 'Python Programming'
>>> a = s.split(' ')
>>> a
['Python', 'Programming']
>>>
>>> a[0]
'Python'
>>> a[1]
'Programming'
>>> s.split('n')
['Pytho', ' Programmi', 'g']
>>> s1 = 'Ibyambyabyboy'
>>> s1.split('by')
['I', 'am', 'a', 'boy']
```

프로그래밍 연습

- 문자열을 입력 받은 다음, 문자열에 있는 문자가 여러 개 있을 경우 처음 나오는 문자 하나만 출력하는 프로그램 작성
- 함수 delDup(x)에서 메소드 count()를 사용
- 프로그램의 실행 예

```
>>>
s = aabbcc
중복이 제거된 문자열 :  abc
>>>
s = bbccabcd
중복이 제거된 문자열 :  bcad
>>>
```

프로그램 작성

- 1개의 단어를 입력하여 이 단어가 회문인지 여부를 판별하는 프로그램 작성

```
>>>
문자열 입력 : 기러기
회문입니다.
>>>
문자열 입력 : Madam
회문입니다.
>>>
문자열 입력 : string
회문이 아닙니다.
>>>
```

프로그래밍 과제

- 띄어쓰기나 문장부호가 포함된 문장에 대해서도 처리가 가능한 프로그램 작성

```
>>>
문자열 입력 : A man, a plan, a canal - Panama!
변환된 문자열 : amanaplanacanalpanama
회문입니다.
>>>
문자열 입력 : 사전 사! 영한사전 사! 영영사전 사! 한영사전 사!
변환된 문자열 : 사전사영한사전사영영사전사한영사전사
회문입니다.
>>>
```

2.2 어구전철 여부 판별

프로그램 개요 (1)

- 두 개의 문자열을 입력 받아 이들이 어구전철인지 여부를 판별하는 프로그램 작성

> - 입력 : 두 개의 문자열 s1, s2
> - 출력 : s1, s2가 어구전철인지 여부

- 어구전철(語句轉綴, anagram)
  - 어구전철은 단어나 문장에 있는 철자를 뒤섞어서 다른 의미를 가지는 단어나 문장을 만드는 것
  - 어구전철인지 여부를 판별할 때에는 띄어쓰기(space)나 쉼표(comma), 느낌표, 물음표, 따옴표와 같은 문장 부호를 무시하고, 영어에서는 대문자와 소문자를 구분하지 않음
  - 한국어 어구전철 : "국왕" → "왕국", "문전박대" → "대박전문", "자살" → "살자“
  - 영어 어구전철 : "Hamlet" → "Amleth", "Rocket Boys" → "October Sky", "O, Draconian devil!" → "Leonardo da Vinci", "Oh, lame saint!" → "The Mona Lisa"

프로그램 개요 (2)

- 단순한 방법
  - s1의 모든 순열(permutation)을 생성하여 하나씩 s2와 비교
  - s1과 s2가 어구전철인 경우 s1의 순열 중에 s2와 동일한 순열이 있을 것이기 때문에 s1과 s2가 어구전철인지 여부를 판별할 수 있음
- 입력한 문자열의 모든 순열을 생성하는 프로그램

```
def perm(a, i, n):
    if i == n - 1:
        for i in range(n): print(a[i], end='')
        print(end=' ')
    else:
        for j in range(i, n):
            a[i], a[j] = a[j], a[i]
            perm(a, i+1, n)
            a[i], a[j] = a[j], a[i]

string = input('문자열 입력 : ')
N = len(string)
s = []
for i in range(N): s.append(string[i])
perm(s, 0, N)
```

프로그램 개요 (3)

- perm() 함수의 수행 과정 (1)

i와 j의 값			출력
1. i=0, j=0, perm(a, 1, 3)			
	1. i=1, j=1, perm(a, 2, 3)		abc
	2. i=1, j=1		
		1. i=1, j=2, perm(a, 2, 3)	acb
		2. i=1, j=2	
2. i=0, j=0			
1. i=0, j=1, perm(a, 1, 3)			
	1. i=1, j=1, perm(a, 2, 3)		bac
	2. i=1, j=1		
		1. i=1, j=2, perm(a, 2, 3)	bca
		2. i=1, j=2	
2. i=0, j=1			

프로그램 개요 (4)

- perm() 함수의 수행 과정 (2)

i와 j의 값			출력
1. i=0, j=2, perm(a, 1, 3)			
	1. i=1, j=1, perm(a, 2, 3)		cba
	2. i=1, j=1		
		1. i=1, j=2, perm(a, 2, 3)	cab
		2. i=1, j=2	
2. i=0, j=2			

- perm() 함수의 호출 횟수
  - n개의 문자에 대한 순열의 개수는 n!
  - 순열의 개수는 n이 증가함에 따라 엄청나게 빠른 속도로 커지게 되는데, n이 10이 되면 10! = 3,628,800개의 순열을 생성하게 됨
  - 따라서 n이 11 이상이 되면 실행시간이 너무 오려 걸려서 사용하기 어려움

프로그램 작성 (1)

- 두 개의 문자열을 입력 받아 어구전철인지 여부를 출력하는 프로그램 작성

```
>>>
첫 번째 문자열 입력 : O, Draconian devil!
두 번째 문자열 입력 : Leonardo da Vinci
어구전철입니다.
>>>
첫 번째 문자열 입력 : aaaabbccc
두 번째 문자열 입력 : cccbbbaaa
어구전철이 아닙니다.
>>>
```

프로그램 작성 (2)

- 문자열 s1을 리스트 list1으로 변환하는 방법
  - for문 사용

```
list1 = []
for i in range(len(s1)):
    list1.append(s1[i])
```

  - list() 함수 사용

```
list1 = list(s1)
```

프로그래밍 과제

- 입력 받은 문자열의 모든 순열을 출력해 주는 프로그램을 사용하여 첫 번째 문자열의 모든 순열과 두 번째 문자열을 차례대로 비교하여 어구전철인지 여부를 판별하는 프로그램 작성

```
>>>
첫 번째 문자열 입력 : aabbbcc
두 번째 문자열 입력 : bbbaacc
어구전철입니다.
>>>
첫 번째 문자열 입력 : aaaabbccc
두 번째 문자열 입력 : cccbbbaaa
어구전철이 아닙니다.
>>>
```

2.3 암호문 만들기

프로그램 개요 (1)

- 영문 소문자로 이루어진 평문과 26 이하의 자연수 K를 입력한 다음, 평문에 있는 문자를 차례대로 하나씩 K번째 뒤에 있는 문자로 변환하여 암호문을 만드는 프로그램 작성

  - 입력 : 평문과 26 이하의 자연수 K
  - 출력 : 암호문

- 암호학에서는 평문을 암호문으로 변환하는 것을 암호화(encipher)라고 부르고, 암호문을 다시 평문으로 변환하는 것을 복호화(decipher)라고 부름

프로그램 개요 (1)

- 카이사르 암호화(Caesar cipher)
  - 평문에 있는 문자가 알파벳의 N번째 문자라면, 이것을 (N+K)번째 문자로 변경하는데, 카이사르는 K = 3을 사용했다고 알려져 있음
- 카이사르 암호화의 예
  - K = 1일 때, 영문 소문자로 이루어진 문자열에 대한 카이사르 암호화의 예를 보이면 다음과 같음
  - 여기서 띄어쓰기(space)는 영문 소문자 'a'보다 하나 앞에 있고, 'z'보다는 하나 뒤에 있는 문자로 생각함
  - 예를 들어, 'z'에 1을 더하면 ' '이 되고, 'z'에 2를 더하면 'a'가 됨

평문 :	a	t	t	a	c	k		s	a	f	e		z	o	n	**e**
암호문 :	b	u	u	b	d	l	a	t	b	g	f	a		p	o	f

파이썬 문법

아스키 코드 (1)

- 컴퓨터에서 문자를 저장하기 위해서는 전세계적으로 통용되는 표준 형식이 필요함
- 아스키 코드
  - 영문자와 특수문자를 표현하기 위해 미국 표준협회(ANSI: American National Standards Institute)에서 제정한 표준 코드 체계를 아스키(ASCII: American Standard Code for Information Interchange) 코드라고 부름
  - 아스키 코드는 7비트로 문자를 표시하므로 $2^7 = 128$개의 영문자와 특수문자를 표현할 수 있음

아스키 코드 (2)

- 아스키 코드표

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

ord() 함수와 chr() 함수

- ord() 함수 : 문자를 아스키 코드로 변환

```
>>> ord(' ')
32
>>> ord('A')
65
>>> ord('Z')
90
>>> ord('a')
97
>>> ord('z')
122
```

- chr() 함수 : 아스키 코드를 문자로 변환

```
>>> chr(32)
' '
>>> chr(65)
'A'
>>> chr(90)
'Z'
>>> chr(97)
'a'
>>> chr(122)
'z'
```

- 문자열 접합 프로그램

```
c = ord('A') - 1
s = ''
for i in range(26):
    c += 1
    s += chr(c)
print(s)
```

프로그래밍 연습

- 숫자로 된 문자열 2개를 입력 받은 다음, 문자열을 자연수로 바꾸어 주는 프로그램 작성
- 자연수인지 확인하기 위해 두 수의 합을 구함
- 함수 makeNum(x)에서 ord() 함수 사용
- 프로그램의 실행 예

```
>>>
a = 123
b = 45
두 수의 합 :  168
>>>
```

프로그램 작성

- 영문 소문자로 이루어진 문자열을 입력 받아 암호문으로 변환하는 프로그램 작성

```
>>>
평문 입력 : attack safe zone
K 값 입력(1~26) : 1
암호문 출력 : [buubdlatbgfa pof]
>>>
평문 입력 : abc xyz
K 값 입력(1~26) : 3
암호문 출력 : [defc ab]
>>>
평문 입력 : xyz zyx
K 값 입력(1~26) : 3
암호문 출력 : [ abcba ]
>>>
```

프로그래밍 과제

- 암호화된 문자열을 복호화하여 다시 평문으로 변환하는 함수 decipher(c)를 작성

```
>>>
평문 입력 : attack safe zone
K 값 입력 : 1
암호문 : [buubdlatbgfa pof]
복호화된 평문 : [attack safe zone]
>>>
평문 입력 : abc xyz
K 값 입력 : 3
암호문 : [defc ab]
복호화된 평문 : [abc xyz]
>>>
```

2.4 문자열 탐색

프로그램 개요

- 입력된 검색어(패턴)가 문자열(텍스트)에서 처음 발견된 위치를 반환하는 프로그램 작성

  - 입력 : 문자열 text와 문자열 pattern
  - 출력 : text에서 pattern이 처음 발견된 위치

- 문자열 탐색(string searching)
  - HWP나 MS Word와 같은 문서 편집기를 사용하여 문서를 작성하다 보면 어떤 단어가 문서의 어느 곳에 있는지 알고 싶은 경우가 있음
  - 이 경우 문서를 텍스트(text)라고 하고, 찾고 싶은 단어를 패턴(pattern)이라고 하며, 텍스트에서 패턴이 어디에 있는지 알아내는 것을 문자열 탐색이라고 함
  - 문자열 탐색은 문서 처리, 패턴 인식, 음성인식, DNA 염기 서열 분석 등에 주로 사용됨

파이썬 문법

문자열 포매팅 (1)

- 문자열 포매팅(string formatting)
  - 숫자 뒤에 '%'가 오면 나머지 연산자로 사용
  - 문자열 뒤에 '%'가 오면 문자열 포매터(string formatter)로 사용
- 문자열 포매터의 종류와 기능

포매터	기능
%d, %x, %o	십진수, 16진수, 8진수 출력 (복소수는 출력 안됨)
%f %.숫자f	실수 출력 표시할 소수점 아래 자리수 명시
%s	문자열 출력
%%	'%' 문자 자체를 출력

문자열 포매팅 (2)

- 십진수 출력

```
>>> x = 'a = %d.'
>>> a = 12
>>> print(x%a)
a = 12.
```

- 두 개 이상의 변수 출력

```
>>> x = 'a = %d, b = %d.'
>>> a = 12
>>> b = 13
>>> print(x%(a, b))
a = 12, b = 13.
```

문자열 포매팅 (3)

- 실수 출력

```
>>> pi = 3.141592
>>> print('pi = %.3f'%pi)
pi = 3.142
```

- 문자열 출력

```
>>> s1 = 'sight'
>>> s2 = 'mind'
>>> print('Out of %s, out of %s.'%(s1, s2))
Out of sight, out of mind.
```

문자열 포매팅 (4)

- 복소수 출력

```
>>> c = 24 + 3.4j
>>> print('c = %s.'%c)
c = (24+3.4j).

>>> print('c = %d.'%c)
Traceback (most recent call last):
File "<pyshell#28>", line 1, in <module>
print('c = %d.'%c)
TypeError: %d format: a number is required, not complex

>>> print('c = %f.'%c)
Traceback (most recent call last):
File "<pyshell#29>", line 1, in <module>
print('c = %f.'%c)
TypeError: can't convert complex to float
```

파일 입출력 (1)

- 파일을 여는 명령문 형식

```
파일객체 = open(파일명, 모드)
```

- 모드의 종류

모드	설명
r	읽기 모드 (기본값)
r+	읽기 + 쓰기 모드
w	쓰기 모드
a	쓰기 + 이어쓰기 모드
t	텍스트 모드 (기본값)
b	이진(binary) 모드

파일 입출력 (2)

- 파일객체의 메소드
  - read() 메소드 : 데이터 전체를 읽어 옴
  - readline() 메소드 : 한 줄만 읽어 옴
  - close() 메소드 : 파일을 닫음

```
파일객체.read()
파일객체.readline()
파일객체.close()
```

파일 입출력 (3)

- 'input.txt' 파일

```
A lover asked his beloved,
Do you love yourself more
than you love me?
The beloved replied,
I have died to myself
and I live for you.
```

- 파일 전체를 읽어 오기

```
fin = open('input.txt', 'r')
text = fin.read()
print('텍스트 : ', text)
fin.close()
```

파일 입출력 (4)

- 파일에서 데이터를 한 줄만 읽어 오기

```
fin = open('input.txt')
line = fin.readline()
print(line, end='')
```

- 파일에서 데이터를 세 줄만 읽어 오기

```
fin = open('input.txt')
for i in range(3):
    line = fin.readline()
    print(line, end='')
```

파일 입출력 (5)

- readline() 메소드를 사용해서 파일에서 데이터를 모두 읽어 오기

```
fin = open('input.txt')
line = fin.readline()
while line:
        print(line, end='')
        line = fin.readline()
```

파일 입출력 (6)

- 데이터를 파일로 출력

```
파일객체 = open(파일명, 'w')
파일객체.write()
```

- 섭씨를 화씨로 변환하여 파일에 쓰기

```
fout = open('output.txt', 'w')
for c in range(0, 51, 10):
    f = int(c*9/5 + 32)
    out = '섭씨 온도 : %d, 화씨 온도 : %d₩n' % (c, f)
    fout.write(out)
fout.close()
```

파일 입출력 (7)

- 'output.txt' 파일에 출력이 제대로 되었는지 확인

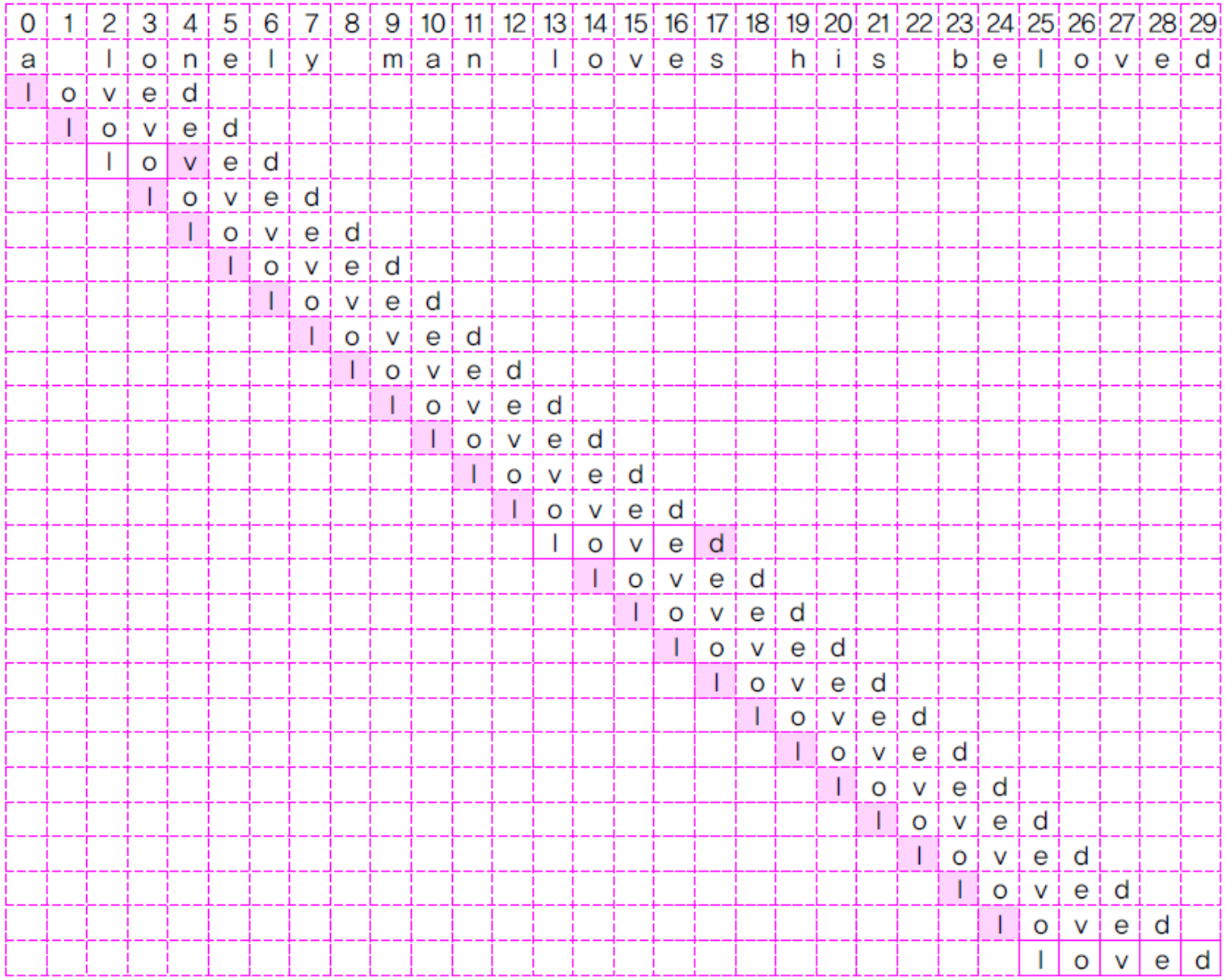
```
fin = open('output.txt')
line = fin.readline()
while line:
    print(line, end='')
    line = fin.readline()
fin.close()
```

프로그래밍 연습

- input.txt에 있는 문장을 입력 받은 다음, 단어별로 끊어서 리스트에 삽입하는 프로그램 작성
- 함수 makeList(x) 사용
- 프로그램의 실행 예
  - input.txt : A lover asked his beloved

```
>>>
['A', 'lover', 'asked', 'his', 'beloved']
>>>
```

패턴 탐색 과정 (1)


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
a l o n e l y m a n l o v e s h i s b e l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d
l o v e d

패턴 탐색 과정 (2)

- 텍스트에서 일치하는 문자가 있는 경우 i와 j의 값

i	…	2	3	4	5	6	7	8	…	12	13	14	15	16	17	18	…	23	24	25	26	27	28	29
텍스트		l	o	n	e	l	y				l	o	v	e	s			b	e	l	o	v	e	d
패턴		l	o	v	e	d					l	o	v	e	d					l	o	v	e	d
j		0	1	2							0	1	2	3	4					0	1	2	3	4

  - "lonely"의 경우 i의 값이 4이고 j의 값이 2인 위치에서 불일치가 발생하였으므로, i의 값 4에서 j의 값 2를 빼서 i에 저장한 다음, i 값에 1을 더한 3번 위치에서 다시 탐색을 시작
  - "loves"의 경우 i의 값이 17이고, j의 값이 4에서 불일치가 발생하였으므로, i의 값 17에서 j의 값 4을 빼서 i에 저장한 다음, i의 값에 1을 더한 14번 위치에서 다시 탐색을 시작
  - "beloved"의 경우에는 i의 값 30에서 j의 값이 5가 되는데, 이렇게 j의 값이 패턴의 길이 M과 같아지게 되면 i-M, 즉, 30에서 5를 뺀 25를 반환

프로그램 작성

- 텍스트와 패턴을 입력하여 텍스트에서 패턴이 처음 발견된 위치를 출력하는 프로그램 작성

```
>>>
텍스트 : A lover asked his beloved, Do you love yourself more than you love me?
패턴 입력 : love
패턴을 처음 발견한 위치 : 2
>>>
텍스트 : A lover asked his beloved, Do you love yourself more than you love me?
패턴 입력 : you
패턴을 처음 발견한 위치 : 30
>>>
텍스트 : A lover asked his beloved, Do you love yourself more than you love me?
패턴 입력 : can
패턴을 발견하지 못함.
>>>
```

프로그래밍 과제

- 텍스트에 패턴이 여러 개 있을 경우 패턴을 발견한 위치를 모두 찾아서 출력해 주는 프로그램 작성

```
>>>
텍스트 : A lover asked his beloved, Do you love yourself more than you love me?
패턴 입력 : love
패턴을 발견한 위치 : 2
패턴을 발견한 위치 : 20
패턴을 발견한 위치 : 34
패턴을 발견한 위치 : 62
문자열 탐색 완료.
>>>
텍스트 : A lover asked his beloved, Do you love yourself more than you love me?
패턴 입력 : you
패턴을 발견한 위치 : 30
패턴을 발견한 위치 : 39
패턴을 발견한 위치 : 58
문자열 탐색 완료.
>>>
```

2.5 최장 부분문자열

프로그램 개요

- 문자열을 입력 받아 중복 문자가 없는 최장 부분문자열(longest substring) 의 길이를 구하는 프로그램 작성

  - 입력 : 문자열 s
  - 출력 : 최장 부분문자열의 길이

- 부분문자열(substring)
  - 문자열에 있는 문자들이 연속적으로 나열되어 있는 문자열의 일부
  - 예) 문자열 "abc"의 부분문자열 : "a", "b", "c", "ab", "bc", "abc“
  - "ac"는 문자 'a'와 'c'가 "abc"에서 연속되어 있지 않으므로 부분문자열이 아님
- 중복 문자가 없는 최장 부분문자열의 예

입력 받은 문자열	최장 부분문자열	길이
abccbadbb	cbad	4
bbbb	b	1

파이썬 문법

수학 관련 내장 함수 (1)

- 수학 관련 내장 함수

함수	설명
abs(x)	x의 절대값을 반환
max(iterable)	매개변수 또는 연속열의 원소 중에서 최대값을 반환
min(iterable)	매개변수 또는 연속열의 원소 중에서 최소값을 반환
round(x [, ndigit])	x를 ndigit 자리에서 반올림한 수를 반환
sum(iterable)	연속열의 원소들의 합을 반환
pow(x, y)	x의 y 제곱을 반환

※ iterable : 왼쪽에서 오른쪽으로 진행하면서 하나씩 원소를 넘겨줄 수 있는 자료형인 리스트, 집합, 튜플, 사전 등을 의미하며, iterable한 자료형을 통틀어서 연속열이라고 부름

수학 관련 내장 함수 (2)

- 수학 관련 내장 함수의 사용 예

```
>>> abs(10)
10
>>> abs(-10)
10
>>> max(2, 3, 5, 1, 4)
5
>>> max([2, 3, 5, 1, 4])
5
>>> min(2, 3, 5, 1, 4)
1
>>> min([2, 3, 5, 1, 4])
1
>>> round(2.6)
3
>>> sum([2, 3, 5, 1, 4])
15
>>> pow(2, 10)
1024
```

수학 관련 내장 함수 (2)

- sum() 함수 : 연속열의 합만을 구하고, 매개변수들의 합을 구하지 않음

```
>>> sum(1,2)
Traceback (most recent call last):
File "<pyshell#48>", line 1, in <module>
sum(1,2)
TypeError: 'int' object is not iterable
>>>
```

- round() 함수

```
>>> round(2.6767, 2)
2.68
>>> round(2.5)
2
>>> round(2.675, 2)
2.67
```

실수 자료형인 float를 처리할 때 근사치로 취급하기 때문에 부정확한 결과를 출력함

math 모듈의 함수 (1)

- math 모듈: 제곱근(sqrt)이나 올림(ceil), 내림(floor), 로그함수, 삼각함수 등을 사용하고 싶은 경우에 사용

```
>>> import math
>>> math.sqrt(2)
1.4142135623730951
>>> math.sqrt(256)
16.0
>>> math.ceil(2.5)
3
>>> math.floor(2.5)
2
>>> math.log(1)
0.0
>>> math.log(math.e)
1.0
>>> math.log10(1000)
3.0
>>> math.log2(1024)
10.0
>>>
```

math 모듈의 함수 (2)

- 각도 변화에 따른 삼각함수표 출력 프로그램

```
import math

print(' x     sin x      cos x      tan x')
print('=====================================')
for i in range(0, 91, 5):
    sinx = math.sin(math.pi * (i / 180))
    cosx = math.cos(math.pi * (i / 180))
    tanx = math.tan(math.pi * (i / 180))
    print("%3d    %0.4f     %0.4f     %0.4f"%(i, sinx, cosx, tanx))
print('=====================================')
```

프로그램 작성

- 문자열을 입력 받아 최장 부분문자열의 길이를 출력하는 프로그램 작성

```
>>>
문자열 입력 : abccbadbb
최장 부분문자열의 길이 : 4
>>>
문자열 입력 : abcd
최장 부분문자열의 길이 : 4
>>>
문자열 입력 : bbbb
최장 부분문자열의 길이 : 1
>>>
```

프로그래밍 과제

- 문자열을 입력 받아 최장 부분문자열을 출력하는 프로그램 작성

```
>>>
문자열 입력 : abccbadbb
최장 부분문자열 : cbad
>>>
문자열 입력 : abcd
최장 부분문자열 : abcd
>>>
문자열 입력 : bbbb
최장 부분문자열 : b
>>>
```