**문자열 string**

* 문자열 text
* 패턴 매칭
* 문자열 암호화
* 문자열 압축

1. **문자의 표현** 
   1. **컴퓨터에서 문자의 표현**

문자 대응 숫자를 정해두고 저장

52 – 6 비트(2^6) 000000

* 1. **표준안 ASCII (아스키 문자) –** *컴퓨터가 문자를 읽고 쓰는데 사용*

차이 값이 같다. 1씩 차례대로 증가하기 때문

Byte 단위 (4byte)

* 1. **유니 코드 --** *다국어 처리 위한 코드*

변수 코드의 크기를 정의

바이트 코드 표준화 할 수 없었다.

* 1. **Big-endian, little-endian** *– 표준 (걸리버 여행기)*

바이트 저장 오더 차이가 있음

1. **문자의 열**
   1. **문자열의 종류**
   2. **참고**
      * List(‘문자열’) : 한 글자씩 str 저장
      * Strlen 함수 구현 해보기
2. def strlen(arr):  
    count = 0  
    while arr[count] = '\0':  
    count += 1  
    return count
3. **파이썬에서 문자열 처리** 
   1. +연결 \*연결
   2. 시퀀스 자료형 , 인덱싱/ 슬라이싱 가능
4. **문자열 뒤집기**
   1. 자기문자열에서 뒤집기 / 새로운 문자열
      * 자기문자열에서 뒤집기

파이썬에 맞지 않음 ,회문을 쓸 때 사용

Swap 1🡪8 2🡪7 …

한번 교환 🡪 두개 변환 : int( n/2 )

mystr = 'algorithm'  
#1. 파이썬 내장 기능  
print(mystr[::-1])  
arr = list(mystr)  
print(arr)  
  
#for  
mystr = 'algorithm'  
arr = list(mystr)  
for i in range(len(arr)//2):  
 arr[i], arr[len(arr)-1-i] = arr[len(arr)-1-i], arr[i]  
print(arr)

* + - 새로운 문자열

1. **파이썬** 
   1. 파이썬은 == , is 연산자 제공
2. s1 = 'abc'  
   s2 = 'abc'  
   s3 = 'def'  
   s4 = s1  
   s5 =s1[:2] + 'c'  
   print(s1 == s2, s1 is s2)  
   print(s1 == s3, s1 is s3)  
   print(s1 == s4, s1 is s4)  
   print(s1 == s5, s1 is s5)

결과 : True True/ False False/ True True/ True False

1. **비교 함수**
   1. 아스키 값으로 비교 해서 ,

값이 크다 = 사전 순서에 뒤진다 = -1 리턴

값이 작다 = 사전 순서에 앞선다 = 1 리턴

문자열 같음 0 리턴

1. **패턴 매칭** 
   * + 처음부터 끝까지 차례대로 순회하면서 문자들을 일일이 비교하는 방식
     + 텍스트에서 패턴이 존재하는 모든 위치를 찾는 문제
     + 텍스트는 t 패턴은 p 로 표현한다.
     + 텍스트의 인덱스 i , 패턴은 j 로 사용한다.
     + 텍스트의 길이 n 패턴은 m으로 쓴다.

비교시 p[j] ==t[i]

1. **패턴 매칭**
   1. 고지식한 패턴 검색알고리즘 (Brure Force)
      * 일치하는 경우

i j 증가 시켜 다음 문자 비교

* + - 불일치하는 경우

다음꺼 준비

* + - 최악의 경우 O(MN)
  1. 카프 라빈 알고리즘
  2. KMP 알고리즘 ???
     + 공통되는 접두어 찾아주기, 옆으로 shift 해준다.
  3. 보이어 무어 알고리즘
* 다양한 문자가 많을 경우에 유리하다 !!
* 텍스트 문자열 한번씩 흝음 !
* 최악 O(mn)

1. **Bit의 암호화**

배타적 논리합 연산 사용

되돌릴때도 마찬가지로 배타적 논리합 사용

1. **문자열 압축**

같은 값이 몇번 반복되는가 ?

AABBBBA

A2B4