Strings

A string can be read from the standard input using the function `input()` or defined in single or double quotes. Two strings can be concatenated, and we can also repeat a string n times multiplying it by integer:

```
print('>_< ' * 5)  # >_< >_< >_< >_< >_<
```

A string in Python is a sequence of characters. The function `len(some_string)` returns how many characters there are in a string:

```
print(len('abcdefghijklmnopqrstuvwxyz'))  # 26
```

Every object in Python can be converted to string using the function `str(some_object)`. So we can convert numbers to strings:

```
s = str(2 ** 100)

print(s)  # 1267650600228229401496703205376

print(len(s))  # 31
```

2.1. Single character

A slice gives from the given string one character or some fragment: substring or subsequence.

There are three forms of slices. The simplest form of the slice: a single character slice `S[i]` gives `i`th character of the string. We count characters starting from 0. That is, if `S = 'Hello'`, `S[0] == 'H'`, `S[1] == 'e'`, `S[2] == 'l'`,`S[3] == 'l'`, `S[4] == 'o'`. Note that in Python there is no separate type for characters of the string. `S[i]` also has the type `str`, just as the source string.

Number `i` in `S[i]` is called an *index*.

If you specify a negative index, then it is counted from the end, starting with the number `-1`. That is, `S[-1] == 'o'`,`S[-2] == 'l'`, `S[-3] == 'l'`, `S[-4] == 'e'`, `S[-5] == 'H'`.

Let's summarize it in the table:

String S	H	e	l	l	o

Index	S[0]	S[1]	S[2]	S[3]	S[4]
Index	S[-5]	S[-4]	S[-3]	S[-2]	S[-1]

If the index in the slice `S[i]` is greater than or equal to `len(S)`, or less than `-len(S)`, the following error is caused `IndexError: string index out of range`.

Substring

Slice with two parameters `S[a:b]` returns the substring of length `b - a`, starting with the character at index `a` and lasting until the character at index b, not including the last one. For example, `S[1:4] == 'ell'`, and you can get the same substring using `S[-4:-1]`. You can mix positive and negative indexes in the same slice, for example, `S[1:-1]`is a substring without the first and the last character of the string (the slice begins with the character with index 1 and ends with an index of -1, not including it).

Slices with two parameters never cause `IndexError`. For example, for `S == 'Hello'` the slice `S[1:5]` returns the string `'ello'`, and the result is the same even if the second index is very large, like `S[1:100]`.

If you omit the second parameter (but preserve the colon), then the slice goes to the end of string. For example, to remove the first character from the string (its index is 0) take the slice `S[1:]`. Similarly if you omit the first parameter, then Python takes the slice from the beginning of the string. That is, to remove the last character from the string, you can use slice `S[:-1]`. The slice `S[:]` matches the string `S` itself.

Immutability of strings

Any slice of a string creates a new string and never modifies the original one. In Python strings are immutable, i.e they can not be changed as the objects. You can only assign the variable to the new string, but the old one stays in memory.

In fact in Python there is no variables. There are only the names that are associated with any objects. You can first associate a name with one object, and then — with another. Can several names be associated with one and the same object.

Let's demonstrate that:

```
s = 'Hello'
```

```
t = s  # s and t point to the same string

t = s[2:4]  # now t points to the new string 'll'

print(s)  # prints 'Hello' as s is not changed

print(t)  # prints 'll'
```

Subsequence

If you specify a slice with three parameters `S[a:b:d]`, the third parameter specifies the step, same as for function`range()`. In this case only the characters with the following index are taken: `a` `a + d`, `a + 2 * d` and so on, until and not including the character with index `b`. If the third parameter equals to 2, the slice takes every second character, and if the step of the slice equals to `-1`, the characters go in reverse order. For example, you can reverse a string like this: `S[::-1]`. Let's see the examples:

```
s = 'abcdefg'

print(s[1])

print(s[-1])

print(s[1:3])

print(s[1:-1])

print(s[:3])

print(s[2:])

print(s[:-1])

print(s[::2])

print(s[1::2])

print(s[::-1])
```

Note how the third parameter of the slice is similar to the third parameter of the function `range()`:

```
s = 'abcdefghijklm'

print(s[0:10:2])
```

```
for i in range(0, 10, 2):
    print(i, s[i])
```

String methods

A method is a function that is bound to the object. When the method is called, the method is applied to the object and does some computations related to it. Methods are invoked as `object_name.method_name(arguments)`. For example, in `s.find("e")` the string method `find()` is applied to the string `s` with one argument `"e"`.

find() and rfind() methods

Method `find()` searches a substring, passed as an argument, inside the string on which it's called. The function returns the index of the first occurrence of the substring. If the substring is not found, the method returns -1.

```
s = 'Hello'

print(s.find('e'))

# 1

print(s.find('ll'))

# 2

print(s.find('L'))

# -1
```

Similarly, the method `rfind()` returns the index of the last occurrence of the substring.

```
s = 'abracadabra'

print(s.find('b'))

# 1

print(s.rfind('b'))

# 8
```

If you call `find()` with three arguments `s.find(substring, left, right)`, the search is performed inside the slice `s[left:right]`. If you specify only two arguments, like `s.find(substring, left)`, the search is performed in the slice `s[left:]`, that is, starting with the character at index `left` to the end of the string. Method `s.find(substring, left, right)` returns the absolute index, relatively to the whole string `s`, and not to the slice.

s = 'my name is bond, james bond, okay?'

print(s.find('bond'))

11

print(s.find('bond', 12))

23

replace() method

Method `replace()` replaces all occurrences of a given substring with another one. Syntax: `s.replace(old, new)` takes the string `s` and replaces all occurrences of substring `old` with the substring `new`. Example:

print('a bar is a bar, essentially'.replace('bar', 'pub'))

# 'a pub is a pub, essentially'

One can pass the third argument `count`, like this: `s.replace(old, new, count)`. It makes `replace()` to replace only first `count` occurrences and then stop.

print('a bar is a bar, essentially'.replace('bar', 'pub', 1))

# 'a pub is a bar, essentially'

count() method

This method counts the number of occurrences of one string within another string. The simplest form is this one: `s.count(substring)`. Only non-overlapping occurrences are taken into account:

print('Abracadabra'.count('a'))

4

print(('aaaaaaaaaa').count('aa'))

5

If you specify three parameters `s.count(substring, left, right)`, the count is performed within the slice `s[left:right]`.

Problem «Slices» (Easy)

Statement

You are given a string.

In the first line, print the third character of this string.

In the second line, print the second to last character of this string.

In the third line, print the first five characters of this string.

In the fourth line, print all but the last two characters of this string.

In the fifth line, print all the characters of this string with even indices (remember indexing starts at 0, so the characters are displayed starting with the first).

In the sixth line, print all the characters of this string with odd indices (i.e. starting with the second character in the string).

In the seventh line, print all the characters of the string in reverse order.

In the eighth line, print every second character of the string in reverse order, starting from the last one.

In the ninth line, print the length of the given string.

Your solution

```
# Read an integer:

s = str(input())

print(s[2])

print(s[-2])

print(s[0:5])

print(s[:-2])
print(s[0::2])

print(s[1::2])

print(s[::-1])

print(s[::-2])

print(len(s))
```

Problem «The number of words» (Easy)

Statement

Given a string consisting of words separated by spaces. Determine how many words it has. To solve the problem, use the method `count`.

Your solution

```
s = str(input())

print(s.count(' ')+1)
```

Suggested solution

```
print(input().count(' ') + 1)
```

Problem «The two halves» (Easy)

Statement

Given a string. Cut it into two "equal" parts (If the length of the string is odd, place the center character in the first string, so that the first string contains one more characther than the second). Now print a new string on a single row with the first and second halfs interchanged (second half first and the first half second)

Don't use the statement `if` in this task.

Your solution

```
s =str(input())

length = len(s)

b = length//2

if length%2 == 0:

   print(s[b:]+s[0:b])

else:

   b=b+1

   print(s[b:]+s[0:b])
```

Suggested solution

```
s = input()

print(s[(len(s) + 1) // 2:] + s[:(len(s) + 1) // 2])
```

Problem «To swap the two words» (Easy)

Statement

Given a string consisting of exactly two words separated by a space. Print a new string with the first and second word positions swapped (the second word is printed first).

This task should not use loops and `if`.

Your solution

```
s = str(input())

a = s.find(" ")

b = a+1

print(s[b:]+" "+ s[0:b])
```

Suggested solution

```
s = input()

first_word = s[:s.find(' ')]

second_word = s[s.find(' ') + 1:]

print(second_word + ' ' + first_word)
```

Problem «The first and last occurrence» (Medium)

Statement

Given a string that may or may not contain a letter of interest. Print the index location of the first and last appearance of `f`. If the letter `f` occurs only once, then output its index. If the letter `f` does not occur, then do not print anything.

Don't use loops in this task.

Your solution

```
s = str(input())

d = s.count("f")

if d ==1:

   print(s.find("f"))

elif d>=2:

   print(str(s.find("f"))+" "+str(s.rfind("f")))
```

Suggested solution

```
s = input()

if s.count('f') == 1:

   print(s.find('f'))

elif s.count('f') >= 2:

   print(s.find('f'), s.rfind('f'))
```

Problem «The second occurrence» (Medium)

Statement

Given a string that may or may not contain the letter of interest. Print the index location of the **second** occurrence of the letter `f`. If the string contains the letter `f` only once, then print `-1`, and if the string does not contain the letter `f`, then print `-2`.

Your solution

```
s = str(input())

if s.find('f') >= 0 and s.find('f') == s.rfind('f'):

    print(-1)

elif s.find('f') != s.rfind('f'):

    n = s.find('f') +1

    print(s.find('f', n))

elif s.find('f') < 0:

    print(-2)

else:

    print(s.rfind('f'))
```

Suggested solution

```
s = input()

if s.count('f') == 1:

    print(-1)

elif s.count('f') < 1:

    print(-2)

else:

    print(s.find('f', s.find('f') + 1))
```

Problem «Remove the fragment» (Medium)

Statement

Given a string in which the letter `h` occurs at least twice. Remove from that string the first and the last occurrence of the letter `h`, as well as all the characters between them.

Your solution

```
s = str(input())

first = s.find('h')

second = s.rfind('h') + 1

d = s[0:first]+ s[second:]

print(d)
```

Suggested solution

```
s = input()

s = s[:s.find('h')] + s[s.rfind('h') + 1:]

print(s)
```

Problem «Reverse the fragment» (Medium)

Statement

Given a string in which the letter `h` occurs at least two times, reverse the sequence of characters enclosed between the first and last appearances.

Your solution

```
s = str(input())

if s.count("h")>=2:

   a=s.find("h")

   b=s.rfind("h")

   a+=1
   c=int(a)

   d=int(b)

   e=s[c:d]

   print(s[:c]+e[::-1]+s[d:])
```

Suggested solution

```
s = input()

a = s[:s.find('h')]

b = s[s.find('h'):s.rfind('h') + 1]

c = s[s.rfind('h') + 1:]

s = a + b[::-1] + c

print(s)
```

Problem «Replace the substring» (Medium)

Statement

Given a string. Replace in this string all the numbers `1` by the word `one`.

Your solution

```
s = str(input())

print(s.replace('1', 'one'))
```

Suggested solution

```
print(input().replace('1', 'one'))
```

Problem «Delete a character» (Medium)

Statement

Given a string. Remove from this string all the characters `@`.

Your solution

```
s = input()

print(s.replace("@", ""))
```

Suggested solution

```
print(input().replace('@', ''))
```

Problem «Replace within the fragment» (Hard)

Statement

Given a string. Replace every occurrence of the letter `h` by the letter `H`, except for the first and the last ones.

Your solution

```
s = input()

c = s[s.find("h")+1: s.rfind("h")]

c=c.replace("h", "H")

print(s[:s.find("h")+1]+c+s[s.rfind("h"):])
```

Suggested solution

```
s = input()

a = s[:s.find('h') + 1]

b = s[s.find('h') + 1:s.rfind('h')]

c = s[s.rfind('h'):]

s = a + b.replace('h', 'H') + c

print(s)
```