Strings and characters

- A string is a sequence of characters. You can access the individual characters in a string using an **index** which starts from zero.

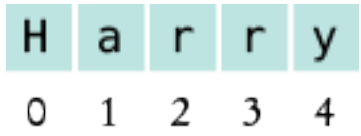

Figure 1: Picture of string

- An individual character is accessed using a special subscript notation For example, if the variable name is defined as:

```
name = "Harry"
```

the statements

```
first = name[0]
last = name[4]
```

- extract the 1st and 5th characters from the string.

String length

- The index value must be within the valid range of character positions or an "index out of range" exception will be generated at run time.
- The `len()` function can be used to determine the position of the last index, or the last character in a string.

```
pos = len(name) - 1    # Length of "Harry" is 5
last = name[pos]       # last is set to "y"
```

Example: indexes and strings

Figure 2: Image of a string

```
##
#  This program prints the initials.
#

first = "Rodolfo"
second = "Sally"

initials = first[0] + "&" + second[0]
print(initials)
```

Negative subscripts

- Python also allows you to count backwards from the end of a list (a string in this case).

```
##
#  This program prints the last letter of the name.
#

# Set up the names.
first = "Rodolfo"
second = "Sally"

# Display the last letters.
ends_with = first[-1] + "&" + second[-1]
print(ends_with)
```

A slice of string

- Sometimes you just want to extract a substring from a string. Python makes this easy with the slice notation:

```
greeting = "Hello World!"

# Just print "Hello"
print(greeting[0:5])
```

Note: this can also be expressed as:

```
print(greeting[:5])
```

- If the start or the end of the slice is empty, python will assume you mean the start or the end of the string.

```
print(greeting[:])
```

What does this print?

In or not in?

- Use the `in` operator to find out whether a substring is in a string:

```
greeting = "H3llo World!"

if '3' in greeting:
    print("The '3' appears in ", greeting)
```

- Reverse the meaning with `not in`:

```
greeting = "H3llo World!"

if '*' not in greeting:
    print("No asterisks in ", greeting)
```

String methods

- A method is like a function but you use it in the context of Object Oriented Programming (OOP). We will learn about OOP later.
- For now, you should understand that a string is an object. String objects have a set of methods which you can use.
- For example, you can apply the `upper` method to any string, like this:

```
name = "John Smith"
uppercaseName = name.upper()    # Sets uppercaseName to "JOHN SMITH"
```

Note that the method name follows the object, and that a dot (.) separates the object and method name.

More string methods

- There is another string method called `lower` that yields the lowercase version of a string:

```
print(name.lower())   # Prints john smith
```

- Just like function calls, method calls can have arguments.
- For example, the string method `replace` creates a new string in which every occurrence of a given substring is replaced with a second string:

```
name2 = name.replace("John", "Jane")   # Sets name2 to
                                        # "Jane Smith"
```

A complete list of string methods can be found in the Python documentation.

Important: strings are immutable

- Note: None of the method calls change the content of the string on which they are invoked.
- After the call `name.upper()`, the `name` variable still holds "John Smith". The method call returns the uppercase version.
- Similarly, the `replace()` method returns a new string with the replacements, without modifying the original.

Question

Write a function called `swap_pairs`. That accepts a string as a paameter and returns that string with each pair adjacent of characters reversed. If the string has an odd number of letters then the last letter is unchanged. For example: the call `swap_pairs('example')` should return `xemalpe`.