# 14-strings-I

March 13, 2022

## 1 Strings

`string` is a python datatype, just like `int`, `bool`, or `list` are datatypes. The values of type `string` are sequences of characters, and they are written in python enclosed in quotes. These can be double or single quotes:

[1]:
```
s0 = "This is a string."
s1 = 'This is also a string.'
```

Python keeps the string *exactly* as you wrote it. That includes spaces and capitalization.

[2]:
```
s = "tHiS is a    WEIRD   strINg... ... .."
print(s)
```

```
tHiS is a    WEIRD   strINg… … ..
```

If you want to have newline characters (i.e. line breaks) in your string, you can write it using *triple* quotes:

[3]:
```
s0 = """This string has a
line break"""

s1 = '''Maybe you want to write something
that
uses
multiple
lines...'''
```

If you do not want to use triple quotes, you can write the newline character itself (in one line), and when the string is printed that character will be transformed into a line break. The newline character is `\n`.

[4]:
```
s = "This is the first line\nThis is the second line"
print(s) # This character  ^^  is the line break.
```

```
This is the first line
This is the second line
```

What if we want to have quotation marks inside our string?

We can *escape* them! *Escaping* means preceeding the character with a backslash (\).

[5]:
```
s_with_quotes = "And then they asked: \"How can we have quotes in strings?\" "
print(s_with_quotes) #    Open quotes ^^                    closing quotes ^^ ^␣
 ↪last one is the closing string
```

```
And then they asked: "How can we have quotes in strings?"
```

### 1.1 Casting

Everything that is **printed** by python is a *string*. So how can we explain the following code?

[6]:
```
L = [1,5,1,1,0]
print(L)
```

```
[1, 5, 1, 1, 0]
```

In reality, what python is doing under the hood is called *casting*. It tranforms a list into a string, and prints the string. *Casting* in programming is the act of changing the type of a value.

Here are some examples.

[7]:
```
# Cast int into float
float(5)
```

[7]:
```
5.0
```

[8]:
```
# Cast float into int
int(5.0)
```

[8]:
```
5
```

[9]:
```
# Casting float into int
int(5.8)
```

[9]:
```
5
```

[10]:
```
# Cast int into string
str(15)
```

[10]:
```
'15'
```

[11]:
```
# Casts list into string
str([1,2,3])
```

[11]:
```
'[1, 2, 3]'
```

[12]:
```
# Cast string into list
list("python")
```

[12]:
```
['p', 'y', 't', 'h', 'o', 'n']
```

[13]:
```
# Cast list into tuple
tuple(["a", "p", "a", "r", "t"])
```

[13]:
```
('a', 'p', 'a', 'r', 't')
```

[14]:
```
# Cast tuple into list
list(('a', 'p', 'a', 'r', 't'))
```

[14]:
```
['a', 'p', 'a', 'r', 't']
```

**But not every casting works!**

[15]:
```
# Cast int into string fails
int("a")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_228647/3614918111.py in <module>
      1 # Cast int into string fails
----> 2 int("a")

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

## 1.2 Strings are (almost) like lists

We can access characters by indexing:

[16]:
```
s = "Principles of Computing"
print("First character:", s[0])
print("Last character:", s[-1])
```

```
First character: P
Last character: g
```

and substrings can be accessed by slicing:

[17]:
```
s = "Principles of Computing"
print("Characters from indices 2 to 6:", s[2:7])
```

```
Characters from indices 2 to 6: incip
```

The number of characters can be obtained using `len`:

[18]:
```
s = "Principles of Computing"
len(s)
```

[18]:
```
23
```

`in` can find substrings:

[19]:
```
s = "Principles of Computing"
"in" in s
```

[19]:
```
True
```

`count` can count the number a substring occurs:

[20]:
```
s = "Principles of Computing"
s.count("in")
```

[20]:
```
2
```

Note that **capitalization is always respected**.

[21]:
```
s = "Principles of Computing"
s.count("comp")
```

[21]:
```
0
```

We can concatenate strings using +. Note that no extra characters (like spaces) are added.

[22]:
```
s1 = "Principles"
s2 = "Computing"
print(s1 + "of" + s2)
```

```
PrinciplesofComputing
```

**BUT THEY CANNOT BE MODIFIED LIKE LISTS** (strings are immutable)

[23]:
```
s = "Principles of Computing"
s[0] = "C"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_228647/96486337.py in <module>
      1 s = "Principles of Computing"
----> 2 s[0] = "C"

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

### 1.3 Looping through strings

Like we do with lists on the range of the length:

[24]:
```
s = "Principles of Computing"
num_is = 0
for i in range(len(s)):
    if s[i] == "i":
```

```
        num_is += 1

num_is
```

[24]: 3

Or on each character:

[25]:
```
s = "Principles of Computing"
num_is = 0
for c in s:
    if c == "i":
        num_is += 1

num_is
```

[25]: 3

### 1.4 Exercise 1

Given a string `s` representing a piece of text, implement the function `words(s)` that returns a list containing all of the words in this text. You should also get rid of the punctuation marks: . , ! ?.

For example, `words("Once upon a time in a land far far away...")` should return:

`["Once", "upon", "a", "time", "in", "a", "land", "far", "far", "away"]`.

[26]:
```
def words(s):
    return []
```

### 1.5 Exercise 2

Given a string `s`, implement the function `mostFrequent(s)` that returns the most frequent character.

For example, `mostFrequent("exercise 2")` should return `"e"`.

[27]:
```
def mostFrequent(s):
    return ""
```

### 1.6 Exercise 3

Implement the function `combiner(s1, s2)` that takes two strings as parameters and combines them, alternating letters, starting with the first letter of the first String, followed by the first letter of the second String, then second letter of first String, etc. The remaining letters of the longer string are then appended to the end of the combination string and this combination string is returned.

For example, `combiner("SaWr", "tras")` should return `"StarWars"`.

[28]:
```
def combiner(s1, s2):
    return ""
```