Strings

رشته‌ها در پایتون یک عبارت متنی را ذخیره می‌کنند. مثلاً اسم یک نفر! رشته‌ها در حقیقت یک دنباله از کاراکترها هستند. خب این یعنی پایتون برای هر شناسه در رشته یک مرتبه در نظر می‌گیرد. منظورم از مرتبه، ترتیب اول، دوم و ... هست. برای مثال وقتی ما یک عبارت متنی مثل "hello" را می‌نویسیم، و می‌گوییم این یک دنباله از حروف است، یعنی این که می‌توانیم، هر حرف را با اندیس گذاری مناسب جدا کنیم. برای مثال حرف h عنصر اول این عبارت است و ... چیزهایی که در این جلسه می‌بینیم:

1.) ساخت رشته‌ها
2.) نمایش
3.) اندیس گذاری و برش حروف
4.) ویژگی‌ها
5.) متدهای رشته‌ها
6.) قالب بندی چاپ!

Creating a String

To create a string in Python you need to use either `single quotes` or `double quotes`. For example:

```
In [2]: # Single word
        'ب'
```

```
Out[2]: 'ب'
```

```
In [2]: # Entire phrase
        'This is also a string'
```

```
Out[2]: 'This is also a string'
```

```
In [3]: # We can also use double quote
        "String built with double quotes"
```

```
Out[3]: 'String built with double quotes'
```

```
In [3]: # Be careful with quotes!
        ' I'm using single quotes, but this will create an error'
```

```
  File "<ipython-input-3-da9a34b3dc31>", line 2
    ' I'm using single quotes, but this will create an error'
      ^
SyntaxError: invalid syntax
```

The reason for the error above is because the single quote in `I'm` stopped the string. You can use combinations of double and single quotes to get the complete statement.

```
In [14]: "Now I'm ready to use the single quotes inside a string!"
```

```
Out[14]: "Now I'm ready to use the single quotes inside a string!"
```

Now let's learn about printing strings!

Printing a String

Using Jupyter notebook with just a string in a cell will automatically output strings, but the correct way to display strings in your output is by using a print function.

```
In [6]: # We can simply declare a string
        'Hello World'
```

```
Out[6]: 'Hello World'
```

```
In [7]: # Note that we can't output multiple strings this way
        'Hello World 1'
        'Hello World 2'
```

```
Out[7]: 'Hello World 2'
```

We can use a print statement to print a string.

```
In [8]: print('Hello World 1')
        print('Hello World 2')
        print('Use \n to print a new line')
        print('\n')
        print('See what I mean?', 'test')
```

```
Hello World 1
Hello World 2
Use
 to print a new line


See what I mean? test
```

```
In [15]: print('this is a test','text2', sep='|', end='\n')
         print('text3')
```

```
this is a test|text2
text3
```

String Basics

We can also use a function called len() to check the length of a string!

```
In [16]: len('Hello World')
```

```
Out[16]: 11
```

دستور len() که درونی پایتون است، می‌تواند طول یک دنباله را برگرداند. در اینجا دنباله‌ای که بحث می‌کنیم، یک رشته است و بنابراین تعداد شناسه‌های درون آن را برمی‌گرداند.

String Indexing

We know strings are a sequence, which means Python can use indexes to call parts of the sequence. Let's learn how this works.

In Python, we use brackets `[]` after an object to call its index. We should also note that indexing starts at 0 for Python. Let's create a new object called `s` and then walk through a few examples of indexing.

```
a = 'H   E    L    L     O'
idx: |   |    |    |     |
     0   1    2    3     4
    -5  -4   -3   -2    -1

a [start:stop:step] # ===> Retrun from start in some steps to stop - 1
```

```
In [1]: # Assign s as a string
        s = 'Hello World'
```

```
In [18]: #Check
         s
Out[18]: 'Hello World'
```

```
In [19]: # Print the object
         print(s)
         Hello World
```

Let's start indexing!

```
In [23]: # Show first element (in this case a letter)
         s[0]
Out[23]: 'H'
```

```
In [24]: s[1]
Out[24]: 'e'
```

```
In [15]: s[2]
Out[15]: 'l'
```

We can use a `:` to perform *slicing* which grabs everything up to a designated point. For example:

```
In [16]: # Grab everything past the first term all the way to the length of s which i
         s len(s)
         s[1:]
Out[16]: 'ello World'
```

```
In [25]: # Note that there is no change to the original s
         s
Out[25]: 'Hello World'
```

```
In [18]: # Grab everything UP TO the 3rd index
         s[:3]

Out[18]: 'Hel'
```

Note the above slicing. Here we're telling Python to grab everything from 0 up to 3. It doesn't include the 3rd index. You'll notice this a lot in Python, where statements and are usually in the context of "up to, but not including".

```
In [19]: #Everything
         s[:]

Out[19]: 'Hello World'
```

We can also use negative indexing to go backwards.

```
In [18]: # Last letter (one index behind 0 so it loops back around)
         s[-1]

Out[18]: 'd'
```

```
In [21]: # Grab everything but the last letter
         s[:-1]

Out[21]: 'Hello Worl'
```

We can also use index and slice notation to grab elements of a sequence by a specified step size (the default is 1). For instance we can use two colons in a row and then a number specifying the frequency to grab elements. For example:

```
In [22]: # Grab everything, but go in steps size of 1
         s[::1]

Out[22]: 'Hello World'
```

```
In [26]: # Grab everything, but go in step sizes of 2
         s[::2]

Out[26]: 'HloWrd'
```

```
In [27]: # We can use this to print a string backwards
         s[::-1]

Out[27]: 'dlroW olleH'
```

String Properties

شما نمی‌توانید یکی از کاراکترهای درون رشته را مستقیم تغییر دهید. بعدا در مثالی یک روش غیر مستقیم را خواهیم دید.

It's important to note that strings have an important property known as **immutability**. This means that once a string is created, the elements within it can not be changed or replaced. For example:

```
In [28]: s

Out[28]: 'Hello World'
```

In [26]:
```
# Let's try to change the first letter to 'x'
s[0] = 'x'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-26-976942677f11> in <module>()
      1 # Let's try to change the first letter to 'x'
----> 2 s[0] = 'x'

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

Notice how the error tells us directly what we can't do, change the item assignment!

Something we *can* do is concatenate strings!

In [27]:
```
s
```

Out[27]:
```
'Hello World'
```

In [29]:
```
# Concatenate strings!
s + ' concatenate me!'
```

Out[29]:
```
'Hello World concatenate me!'
```

In [2]:
```
1 + s
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-a2d64d34e5a8> in <module>()
----> 1 1 + s

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

In [3]:
```
s + 1
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-b8bba336fa32> in <module>()
----> 1 s + 1

TypeError: can only concatenate str (not "int") to str
```

In [32]:
```
# We can reassign s completely though!
s = s + ' concatenate me!'
```

In [33]:
```
print(s)
```

```
Hello World concatenate me!
```

In [31]:
```
s
```

Out[31]:
```
'Hello World concatenate me!'
```

We can use the multiplication symbol to create repetition!

In [34]:
```
letter = 'zw'
```

In [35]:
```
letter*10
```

Out[35]:
```
'zwzwzwzwzwzwzwzwzwzw'
```

Basic Built-in String methods

Objects in Python usually have built-in methods. These methods are functions inside the object (we will learn about these in much more depth later) that can perform actions or commands on the object itself.

We call methods with a dot, ".", and then the method name. Methods are in the form:

object.method(parameters)

Where parameters are extra arguments we can pass into the method. Don't worry if the details don't make 100% sense right now. Later on we will be creating our own objects and functions!

Here are some examples of built-in methods in strings:

```
In [36]: s

Out[36]: 'Hello World concatenate me!'
```

```
In [26]: # Upper Case a string
         s.upper()

Out[26]: 'HELLO WORLD CONCATENATE ME!'
```

```
In [27]: s

Out[27]: 'Hello World concatenate me!'
```

```
In [36]: # Lower case
         s.lower()

Out[36]: 'hello world concatenate me!'
```

```
In [30]: # Split a string by blank space (this is the default)
         s.split()

Out[30]: ['Hello', 'World', 'concatenate', 'me!']
```

```
In [38]: # Split by a specific element (doesn't include the element that was split o
         n)
         s.split('W')

Out[38]: ['Hello ', 'orld concatenate me!']
```

There are many more methods than the ones covered here. Visit the Advanced String section to find out more!

Print Formatting

We can use the .format() method to add formatted objects to printed string statements.

The easiest way to show this is through an example:

```
In [37]: 'Insert another string with curly brackets: {}'.format('The inserted string
         ')

Out[37]: 'Insert another string with curly brackets: The inserted string'
```

We will revisit this string formatting topic in later sections when we are building our projects!