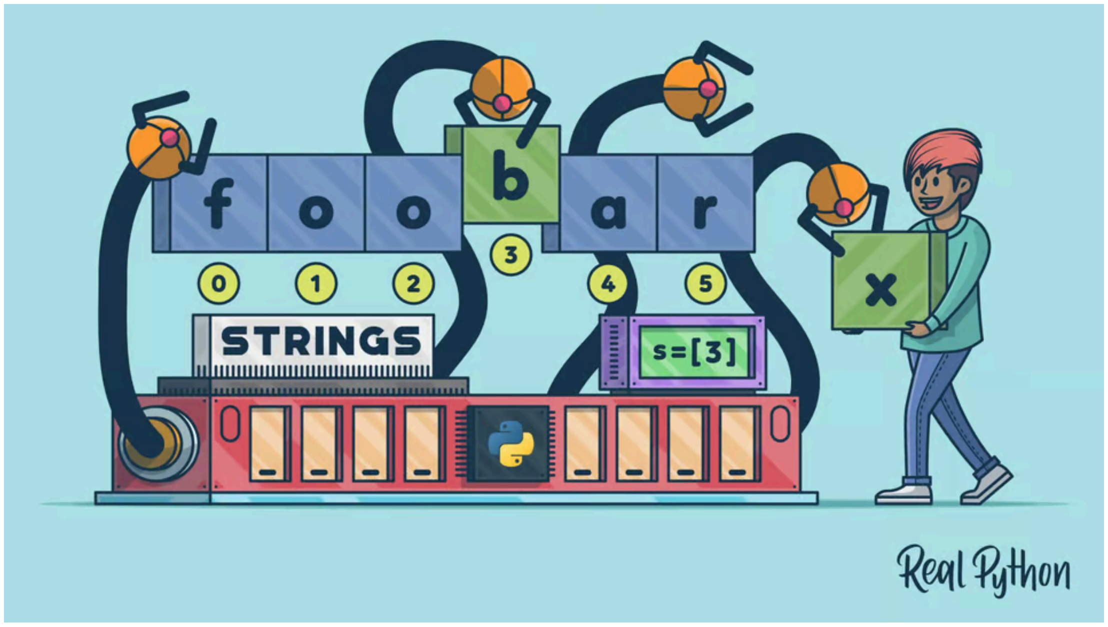
2. String

Get ready to master strings in Python! This section will teach you how to create, manipulate, and work with text data effectively.

Strings are used to store text data in Python. They're considered sequences, meaning Python keeps track of each character in order. This allows us to access specific characters using indexing. Strings are like ordered lists of characters in Python. This means we can refer to individual characters by their position.

We'll learn about the following topics:

- 2.1. Creating Strings
- 2.2. Printing Strings
- 2.3. Built-in String Functions
- 2.4. String Indexing
- 2.5. String Properties
- 2.6. String Operators
- 2.7. Built-in String Methods
- 2.8. String Formatting



Name	Type in Python	Description	Example
Strings	str	Ordered sequence of characters, using the syntax of either single quotes or double quotes.	'hello' "Hello" "i don't do that" "2000f"

In [1]:
```
type('hello')
```

Out[1]:
```
str
```

2.1. Creating Strings:

Strings in Python are created by enclosing text within either single or double quotation marks. This allows Python to differentiate between text and code.

In [2]:
```
#Single Quote
'hello'
```

Out[2]:
```
'hello'
```

In [3]:
```
#phrase
'beauty is in the eye of the beholder'
```

Out[3]:
```
'beauty is in the eye of the beholder'
```

In [4]:
```
#Double Quotes
"beauty is in the eye of the beholder"
```

Out[4]:
```
'beauty is in the eye of the beholder'
```

Please note that if you use ' you also have to use `'` at the end of string not `"` . Moreover, if your text contains `'` in itself, you have to use `"` for determining the string.

In [5]:
```
"I'm happy"
```

Out[5]:
```
"I'm happy"
```

In [6]:
```
'I'm happy'
```

```
  File "C:\Users\Dear User\AppData\Local\Temp\ipykernel_4676\3217421119.py", line 1
    'I'm happy'
       ^
SyntaxError: invalid syntax
```

This error occurred because `'` in the I'm stopped the string. Python will treat the second `'` as an ending character.

2.2. Printing Strings:

In Jupyter Notebook, strings are automatically displayed when you enter them into a cell. However, the proper way to print strings in your output is to use the `print()` function.

Note that we can't output multiple strings without print function. only the last string in a cell will be displayed.

```
In [8]:  'Hello World 1'
          'Hello World 2'

Out[8]:  'Hello World 2'
```

We should use a print statement to print a string.

```
In [9]:  #Using Print
         print('Hello World 1')
         print('Hello World 2')

         Hello World 1
         Hello World 2
```

Special Escape Character	Result
\n	New Line
\t	Tab

```
In [10]:  #\t
          print('beauty \t is in the eye of the beholder')

          beauty 	 is in the eye of the beholder
```

2.3. Built-in String Functions:

`len()`

Python's built-in len() function counts all of the characters in the string, including spaces and punctuation.

```
In [11]:  len('Hello World')

Out[11]:  11
```

`str()`

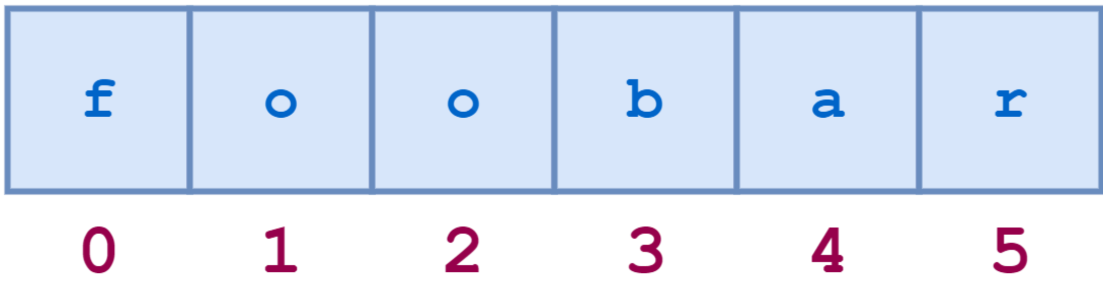
Returns a string representation of an object.

```
In [12]:  str(25.6)

Out[12]:  '25.6'
```

2.4. String Indexing:

Strings are sequences, so we can access specific parts using indexes. Python uses square brackets `[]` for indexing, and it starts from **0**.



In [13]:
```
# Assign a as a string
a = 'beauty is in the eye of the beholder'

a
```

Out[13]:
```
'beauty is in the eye of the beholder'
```

In [14]:
```
#Print the object
print(a)
```

```
beauty is in the eye of the beholder
```

In general, if you want nth character of a string `name_of_string[n-1]`

In [15]:
```
#Show first element
a[0]
```

Out[15]:
```
'b'
```

In [16]:
```
#Show second element
a[1]
```

Out[16]:
```
'e'
```

To get a portion of a string, we can use slicing with the colon `:` .

In general if you want to get everything from nth index to mth index use `name_of_string[n:m+1]`

```
In [17]: a[2:7]

Out[17]: 'auty '
```

```
In [18]: #Grab everything past the first term all the way to the length of a which is len(a)
         a[1:]

Out[18]: 'eauty is in the eye of the beholder'
```

```
In [19]: #Grab every characters up to the 3rd index
         a[:3]

Out[19]: 'bea'
```

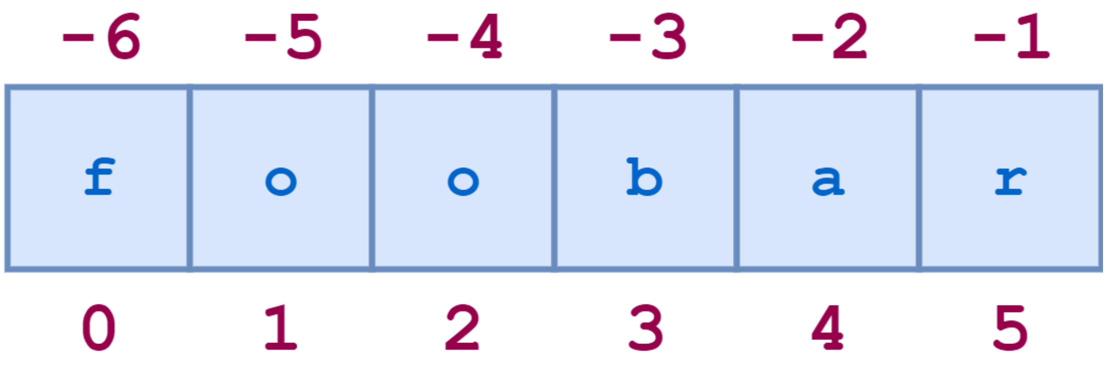
```
In [20]: #Grab Everything
         a[:]

Out[20]: 'beauty is in the eye of the beholder'
```

```
In [21]: a[len(a)-1]

Out[21]: 'r'
```

String indices can also be specified with negative numbers, in which case indexing occurs from the end of the string backward: -1 refers to the last character, -2 the second-to-last character, and so on.



```
In [22]: #Last Chatacter (one index behind 0 so it loops back around)
         a[-1]

Out[22]: 'r'

In [23]: a[-3]

Out[23]: 'd'

In [24]: a[-len(a)]

Out[24]: 'b'

In [25]: #Grab everything but the last Character
         a[:-1]

Out[25]: 'beauty is in the eye of the beholde'

In [26]: a

Out[26]: 'beauty is in the eye of the beholder'

In [27]: print(a[-5:-2])
         print(a[20:23])
         a[-5:-2] == a[20:23]

         old
          of
Out[27]: False
```

Slicing can also have a third elememnt.

`[start:stop:step]`

- start: numerical index for the slice index
- stop: index you will go up to but not include
- step: the size of jump you take

```
In [28]: a[::1]

Out[28]: 'beauty is in the eye of the beholder'

In [29]: #Grab everything, but go in step sizes of 2
         a[::2]

Out[29]: 'bat si h y ftebhle'
```

You can specify a negative step value as well, in which case Python steps backward through the string. In that case, the starting/first index should be greater than the ending/second index.

```
In [30]: a[6:0:-2]
```

```
Out[30]: ' ta'
```

This is a common paradigm for reversing a string.

```
In [31]: a[::-1]
```

```
Out[31]: 'redloheb eht fo eye eht ni si ytuaeb'
```

2.5. String Properties:

- **immutability**: Strings in Python are immutable, meaning you can't change individual characters within them once they're created.

```
In [32]: #Let's try to change the first letter to 'c'
         a[0] = 'c'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_4676\3686050719.py in <module>
      1 #Let's try to change the first letter to 'c'
----> 2 a[0] = 'c'

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

In truth, there really isn’t much need to modify strings. You can usually easily accomplish what you want by generating a copy of the original string that has the desired change in place. There are very many ways to do this in Python. Here is one possibility:

```
In [33]: 'c' + a[1:]
```

```
Out[33]: 'ceauty is in the eye of the beholder'
```

- **concatenate**: Concatenation refers to the process of joining two or more strings together to form a new, longer string. In Python, you can concatenate strings using the addition operator `+`.

```
In [34]: a + a
```

```
Out[34]: 'beauty is in the eye of the beholderbeauty is in the eye of the beholder'
```

```
In [35]: a + ' ' + a
```

```
Out[35]: 'beauty is in the eye of the beholder beauty is in the eye of the beholder'
```

```
In [36]: a + ' New Sentence'
```

```
Out[36]: 'beauty is in the eye of the beholder New Sentence'
```

```
In [37]: a[:3] + a[3:]
```

```
Out[37]: 'beauty is in the eye of the beholder'
```

```
In [38]: a
```

```
Out[38]: 'beauty is in the eye of the beholder'
```

The original a string is unchanged until you reassign a.

- **Reassignment**

```
In [39]: a = a + ' New Sentence!'
```

```
In [40]: print(a)
```

```
beauty is in the eye of the beholder New Sentence!
```

- **Repetition**: In Python, strings support the repetition property, which allows you to create new strings by repeating an existing string multiple times. This is achieved using the multiplication operator `*`.

```
In [41]: a * 2
```

```
Out[41]: 'beauty is in the eye of the beholder New Sentence!beauty is in the eye of the beholde
         r New Sentence!'
```

2.6. String Operators:

- **`in`** : Python also provides a membership operator that can be used with strings. The in operator returns True if the first operand is contained within the second, and False otherwise.

```
In [42]: 'W' in a
```

```
Out[42]: False
```

```
In [43]: 'b' in a
```

```
Out[43]: True
```

- **`not in`** : The not in operator returns True if the first operand is not contained within the second, and False otherwise.

```
In [44]: 'W' not in a
```

```
Out[44]: True
```

```
In [45]: 'b' not in a
```

```
Out[45]: False
```

There are also comparison operators which will be discussed later.

2.7. Built-in String Methods:

Objects in Python have built-in functions called 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 period and then the method name. Methods are in the form:

```
object.method(parameters)
```

Where parameters are extra arguments we can pass into the method.

Method	**Description**
upper()	upper case a string
lower()	lower case a string
split()	Split a string by blank space
split(m)	Split a string by the element m (doesn't include the element that was split on)
replace(oldvalue,newvalue,count(optional))	replaces a specified phrase with another specified phrase
capitalize()	first character converted to uppercase and all other characters converted to lowercase
swapcase()	uppercase alphabetic characters converted to lowercase and vice versa
title()	first letter of each word is converted to uppercase and remaining letters are lowercase
count(m,a(optional),b(optional))	number of occurrences of m within the substring indicated by a and b
endswith(m,a(optional),b(optional))	returns True if an string ends with m within the substring indicated by a and b
find(m,a(optional),b(optional))	see if a string contains a m within the substring indicated by a and b and returns the lowest index
rfind(m,a(optional),b(optional))	see if a string contains a m starting at the end within the substring indicated by a and b and returns the lowest index
index(m,a(optional),b(optional))	the index of first occurrence of m for a given substring indicated by a and b
rindex(m,a(optional),b(optional))	the index of first occurrence of m starting at the end for a given substring indicated by a and b

```
In [46]: a
```

```
Out[46]: 'beauty is in the eye of the beholder New Sentence!'
```

```
In [47]: a.upper()

Out[47]: 'BEAUTY IS IN THE EYE OF THE BEHOLDER NEW SENTENCE!'

In [48]: a.lower()

Out[48]: 'beauty is in the eye of the beholder new sentence!'

In [49]: a.split()

Out[49]: ['beauty',
          'is',
          'in',
          'the',
          'eye',
          'of',
          'the',
          'beholder',
          'New',
          'Sentence!']

In [50]: a.split('b')

Out[50]: ['', 'eauty is in the eye of the ', 'eholder New Sentence!']

In [51]: a

Out[51]: 'beauty is in the eye of the beholder New Sentence!'

In [52]: a.replace('b','c')

Out[52]: 'ceauty is in the eye of the ceholder New Sentence!'

In [53]: a.replace('b','c',1)

Out[53]: 'ceauty is in the eye of the beholder New Sentence!'

In [54]: a.capitalize()

Out[54]: 'Beauty is in the eye of the beholder new sentence!'

In [55]: a.swapcase()

Out[55]: 'BEAUTY IS IN THE EYE OF THE BEHOLDER nEW sENTENCE!'

In [56]: a.title()

Out[56]: 'Beauty Is In The Eye Of The Beholder New Sentence!'

In [57]: a.count('b')

Out[57]: 2
```

```
In [58]: a

Out[58]: 'beauty is in the eye of the beholder New Sentence!'

In [59]: a.count('b',0,3)

Out[59]: 1

In [60]: a.endswith('!')

Out[60]: True

In [61]: a.endswith('!',0,3)

Out[61]: False

In [62]: a.find('Sentence!')

Out[62]: 41

In [63]: a.find('b')

Out[63]: 0

In [64]: a.rfind('b')

Out[64]: 28

In [65]: a.index('b')

Out[65]: 0

In [66]: a.rindex('b')

Out[66]: 28
```

`.rindex()` is identical to `.rfind()`, except that it raises an exception if m is not found rather than returning -1.

```
In [67]: a.rindex('x')

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_4676\1048334928.py in <module>
----> 1 a.rindex('x')

ValueError: substring not found

In [68]: a.rfind('x')

Out[68]: -1
```

2.8. String Formatting:

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

```
In [69]: 'Insert another string with curly brackets: {}'.format('The inserted string')
```

```
Out[69]: 'Insert another string with curly brackets: The inserted string'
```

f-string

```
In [70]: n = 23
         m = 25
         prod = n * m
         print(f'The product of {n} and {m} is {prod}')
```

```
The product of 23 and 25 is 575
```