Strings in Python

String Literals

String literals , or “ strings ,” represent a sequence of characters.

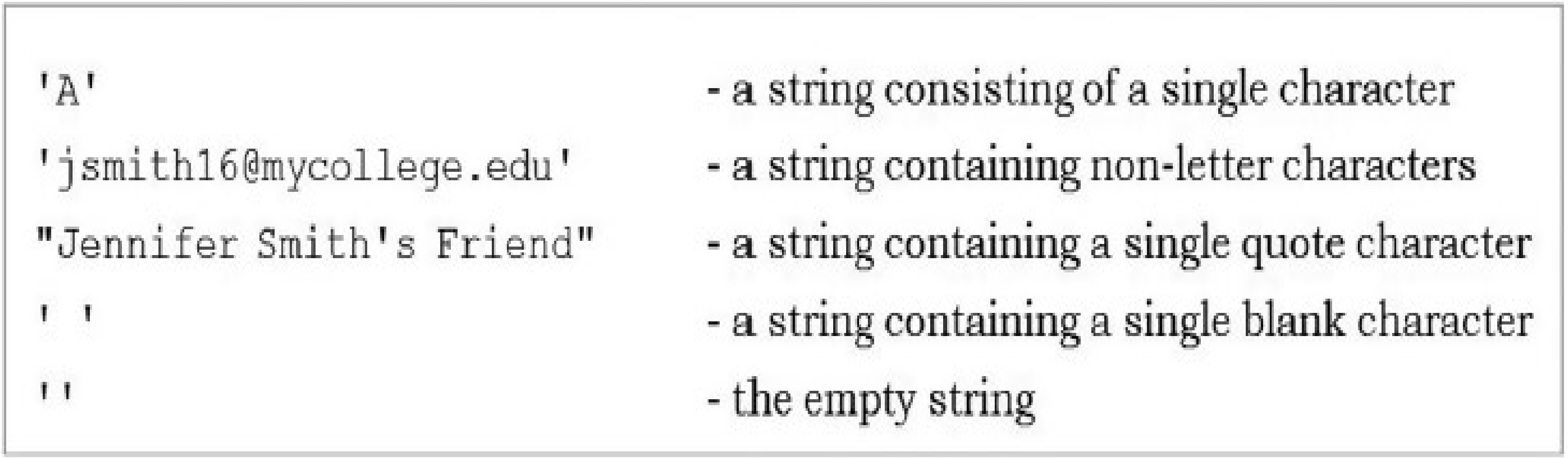
```
'Hello'   'Smith, John'   "Baltimore, Maryland 21210"
```

In Python, Strings can be single (') **or double** (") or **tripple**(“””) **quoted**. Strings must be on one line (except when delimited by triple quotes, discussed later).

```
>>> print('Welcome to Python!')
>>> print(“let’s learn python”)
>>> print(“””Welcome
                      to
                          Python”””)
```

A string may contain zero or more characters, including **letters**, **digits**, **special characters**, and **blanks**. A string consisting of only a pair of matching quotes (with nothing in between) is called the **empty string**, which is different from a string containing only blank characters. Both blank strings and the empty string have their uses, as we will see.

Strings may also contain quote characters as long as different quotes (single vs. double) are used to delimit the string.



`'A'`	- a string consisting of a single character
`'jsmith16@mycollege.edu'`	- a string containing non-letter characters
`"Jennifer Smith's Friend"`	- a string containing a single quote character
`' '`	- a string containing a single blank character
`''`	- the empty string

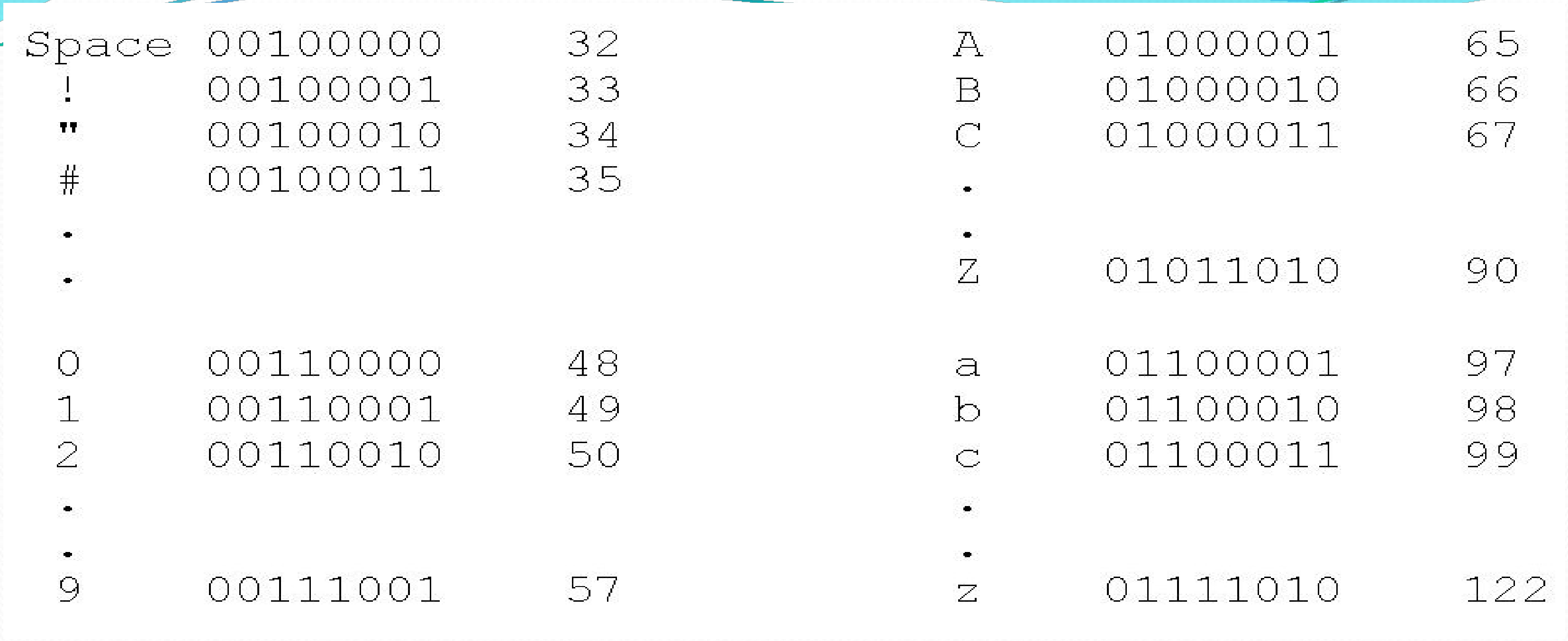
The Representation of Character Values

- **There needs to be a way to encode (represent) characters within a computer**. Although various encoding schemes have been developed, the **Unicode encoding scheme** is intended to be a universal encoding scheme.

- Unicode is actually a collection of different encoding schemes utilizing between 8 and 32 bits for each character. The default encoding in Python uses **UTF-8**, an 8-bit encoding compatible with **ASCII**, an older, still widely used encoding scheme.

- **Currently, there are over 100,000 Unicode-defined characters for many of the languages around the world**. Unicode is capable of defining more than four billion characters. Thus, all the world's languages, both past and present, can potentially be encoded within Unicode.

UTF :Unicode Transformation Format.



Space	00100000	32	A	01000001	65
!	00100001	33	B	01000010	66
"	00100010	34	C	01000011	67
#	00100011	35	.		
.			.		
.			Z	01011010	90
0	00110000	48	a	01100001	97
1	00110001	49	b	01100010	98
2	00110010	50	c	01100011	99
.			.		
.			.		
9	00111001	57	z	01111010	122

Partial listing of the ASCII-compatible UTF-8 encoding scheme

UTF-8 encodes characters that have an ordering with sequential numerical values. For example, 'A' is encoded as 01000001 (65), 'B' is encoded as 01000010 (66), and so on. This is also true for digit characters, ‘0’ is encoded as 48, ‘1’ as 49, etc.

Converting Between a Character and Its Encoding

Python has a means of converting between a character and its encoding.

The ord function gives the UTF-8 (ASCII) encoding of a given character. For example,

`ord('A')` is 65

The chr function gives the character for a given encoding value, thus

`chr(65)` is 'A'

While in general there is no need to know the specific encoding of a given character, there are times when such knowledge can be useful.

Creating Strings in Python

Single Quoted

str=‘Bangalore is a garden city’

Double Quoted

str=“Bangalore is a garden city”

Triple Quoted

str=‘”Bangalore is a garden city’”

Creating a string in Python

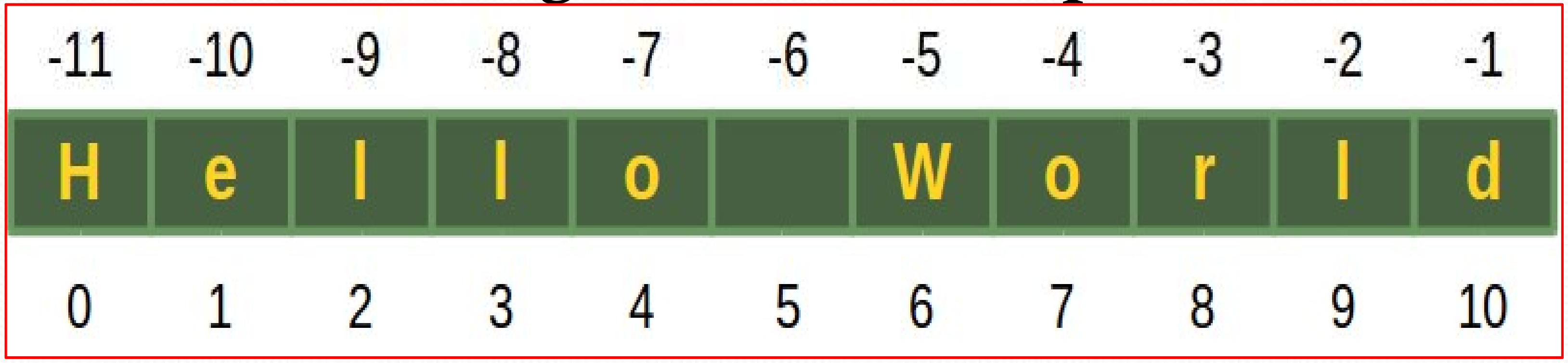
str="BANGALORE”

B	A	N	G	A	L	O	R	E
0	1	2	3	4	5	6	7	8

Accessing string elements
Str[0] returns B
Str[1] returns A

Indexing in string

- A string in Python consists of a series or sequence of characters - letters, numbers, and special characters. Strings can be subscripted or indexed.



-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

s = "Hello World“

s[o] is H

String functions

method	Description
String count()	returns occurrences of substring in string
String format()	formats string into nicer output
String index()	Returns Index of Substring
String islower()	Checks if all Alphabets in a String are Lowercase
String join()	Returns a Concatenated String
String lower()	returns lowercased string
String upper()	returns uppercased string
String strip()	Removes Both Leading and Trailing Characters
String replace()	Replaces Substring Inside
String split()	Splits String from Left
float()	returns floating point number from number, string
input()	reads and returns a line of string
int()	returns integer from a number or string
len()	Returns Length of an Object
max()	returns largest element
min()	returns smallest element
ord()	returns Unicode code point for Unicode character
sorted()	returns sorted list from a given iterable

String functions

```
mystr="Bangalore is a garden city"

print("The orginal string "+mystr)
print(mystr.upper())
print(mystr.lower())
print(mystr[3:6])      # substring or slicing
print(len(mystr))
print(mystr.replace("lore","lure"))
print(mystr.split(" "))
print("The orginal string "+mystr)

print(mystr.count("a",0,len(mystr)))

#str[1]='e'    #error


msg="Capital of Karanataka"
print(mystr+msg)     #concatenation

if("garden" in mystr):
   print("yes garden city")

del mystr
print(mystr)
```

Input() function

- input()
- reads and returns a line of string

Example:

```
data=input(“Enter some data”);
print(data)
Print(data[0])
Print(data[1])
Print(data[2])
```

How to change or delete a string?

mystr="Bangalore is a garden city“

#mystr[o]=‘M'   #error

strings are immutable

But deleting the string entirely is possible

Use the keyword del

del mystr

print(mystr)

Python strings are "immutable" ,they can’t be changed after they are created

String Membership Test

```
str="Bangalore is a garden city"
if("garden" in str):
   print("yes found")
else:
  print(“not found”)
```

String functions

method	Description
String count()	returns occurrences of substring in string
String format()	formats string into nicer output
String index()	Returns Index of Substring
String islower()	Checks if all Alphabets in a String are Lowercase
String join()	Returns a Concatenated String
String lower()	returns lowercased string
String upper()	returns uppercased string
String strip()	Removes Both Leading and Trailing Characters
String replace()	Replaces Substring Inside
String split()	Splits String from Left
float()	returns floating point number from number, string
input()	reads and returns a line of string
int()	returns integer from a number or string
len()	Returns Length of an Object
max()	returns largest element
min()	returns smallest element
ord()	returns Unicode code point for Unicode character
sorted()	returns sorted list from a given iterable

Python String join()

Join all items of a sequence into a string, using a string separator

Example:

```
date=["15","Aug","1947"]

char='-'

print(char.join(date))

15-Aug-1947
```

Python String join()

join() Parameters

Join all items of a sequence into a string, using a string separator:

The join() method takes an iterable

The join() method takes all items in an iterable and joins them into one string.

A string must be specified as the separator.

Syntax:

string.join(*iterable*)

Example 2: split and join

```
mystr="Bangalore is a garden city"

a=mystr.split(" ")

msg=" "

print(msg.join(a))
```

Control Characters

Control characters are special characters that are not displayed, but rather *control* the display of output, among other things. Control characters do not have a corresponding keyboard character, and thus are represented by a combination of characters called an ***escape sequence*** .

Escape sequences begin with an ***escape character*** that causes the characters following it to "escape" their normal meaning. **The backslash** (\\) **serves as the escape character in Python**. For example, **'\n'**, represents the ***newline control character***, that begins a new screen line,

```
print('Hello\nJennifer Smith')
```

which is displayed as follows:

```
Hello
Jennifer Smith
```

backslash notation	Description
\a	Bell or alert
\b	Backspace
\n	Newline
\s	Space
\t	Tab
\\	\ in output
\'	‘ in output
\"	“ in output

Implicit and Explicit Line Joining

Sometimes a program line may be too long to fit in the Python-recommended maximum length of 79 characters. There are two ways in Python to deal with such situations:

- **explicit line joining**
- **implicit line joining**

Explicit Line Joining

program lines may be explicitly joined by use of the backslash (\) **character.** Program lines that end with a backslash that are not part of a literal string (that is, within quotes) continue on the following line.

Error:

```
print('This program will calculate a restaurant tab for a couple
       with a gift certificate, and a restaurant tax of 3%')
```

No Error: **Explicit Line Joining**

```
print('This program will calculate a restaurant tab for a couple \
       with a gift certificate, and a restaurant tax of 3%')
```

Implicit line joining

Expressions in parentheses, square brackets or curly braces can be split over more than one physical line without using backslashes.

For example:

```
month_names=['January', 'Feb', 'March',
                    'April', 'May', 'June', 'July', 'August', 'September',
                    'October', 'November', 'December']
print(month_names)
```

Implicit Line Joining

There are certain delimiting characters that allow a *logical program line* to span more than one *physical line*. This includes matching parentheses, square brackets, curly braces, and triple quotes.

For example, the following two program lines are treated as one logical line:

```
print('Name:',student_name, 'Address:', student_address,
      'Number of Credits:', total_credits, 'GPA:', current_gpa)
```

Matching quotes (except for triple quotes) must be on the same physical line.

```
print('This program will calculate a restaurant tab for a couple’
       ‘with a gift certificate, and a restaurant tax of 3%')
```

String formatting

```
name="Adam"
bal=14500.45

print(name,bal)
```

How to get formatted(nice) output

```
Hello Adam, your balance is 14500.45 cr
```

String formatting

The string format() method formats the given string into a nicer output in Python.

String format() Parameters

- format() method takes any number of parameters. But, is divided into two types of parameters:
- **Positional parameters** - list of parameters that can be accessed with index of parameter inside curly braces {index}
- **Keyword parameters** - list of parameters of type key=value, that can be accessed with key of parameter inside curly braces {key}

Return value from String format()

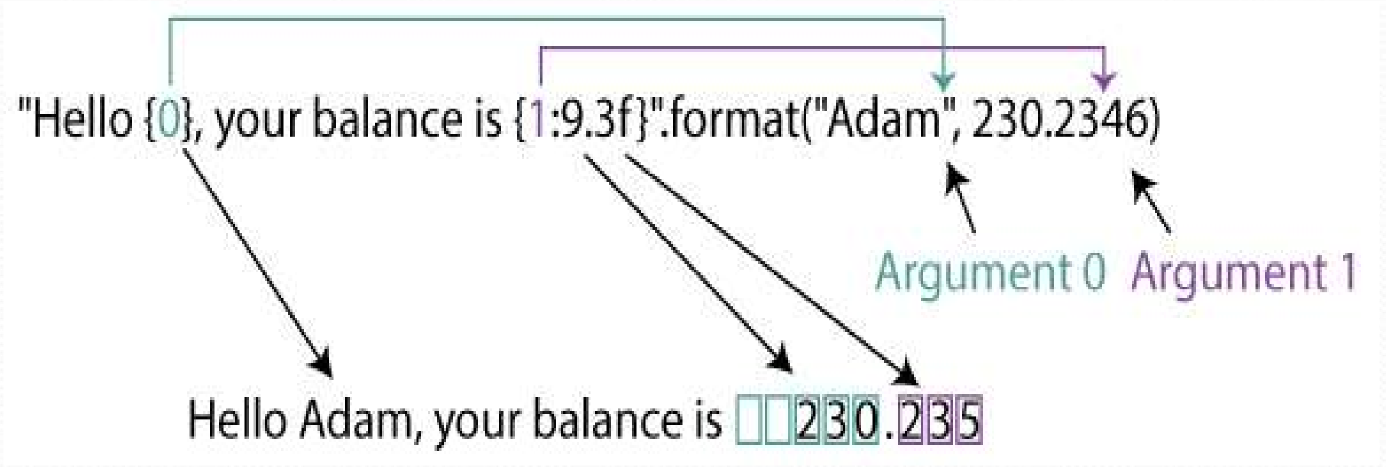
The format() method returns the formatted string.

String formatting

```
name="Adam"
bal=14500.45
print(name,bal)
# default arguments
str="Hello {}, your balance is {}"
print(str.upper())
print(str.format(name,bal))

# default arguments
print("Hello {}, your balance is {}".format(name,bal))

# positional arguments
print("Hello {0}, your balance is {1}".format(name,bal))

# keyword arguments
print("Hello {custname}, your balance is {blc}".format(custname=name, blc=bal))

# mixed arguments
print("Hello {0}, your balance is {blc}".format(name,blc=bal))
```

('curly braces') are used to indicate a replacement field within the string:

How String format() works?
For positional arguments


"Hello {0}, your balance is {1:9.3f}".format("Adam", 230.2346)
Argument 0
Argument 1
Hello Adam, your balance is 230.235

Here, Argument 0 is a string "Adam" and Argument 1 is a floating number 230.2346.

Note: Argument list starts from 0 in Python.

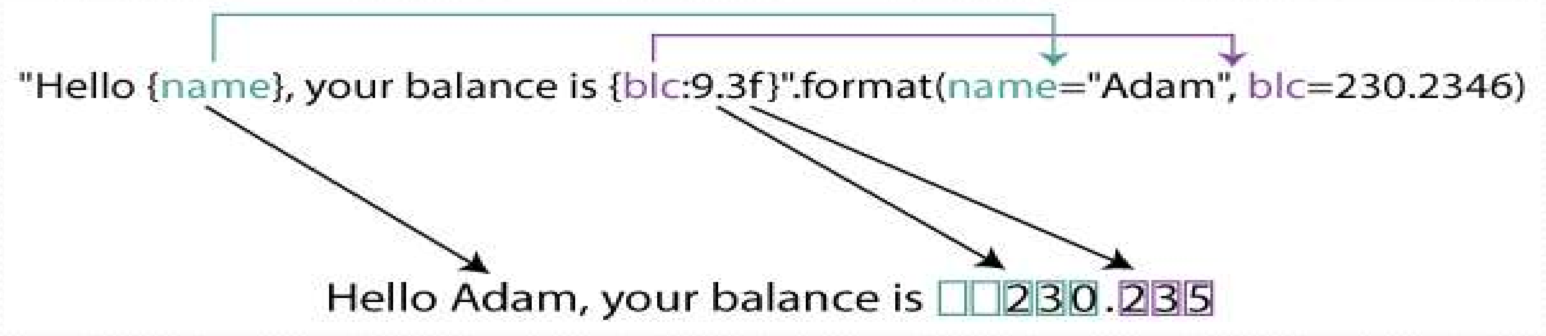
The string `"Hello {0}, your balance is {1:9.3f}"` is the template string.

This contains the format codes for formatting.

The curly braces are just placeholders for the arguments to be placed.

In the above example, `{0}` is placeholder for `"Adam"` and `{1:9.3f}` is placeholder for `230.2346`.

How String format() works?
For keyword arguments



```
"Hello {name}, your balance is {blc:9.3f}".format(name="Adam", blc=230.2346)
```

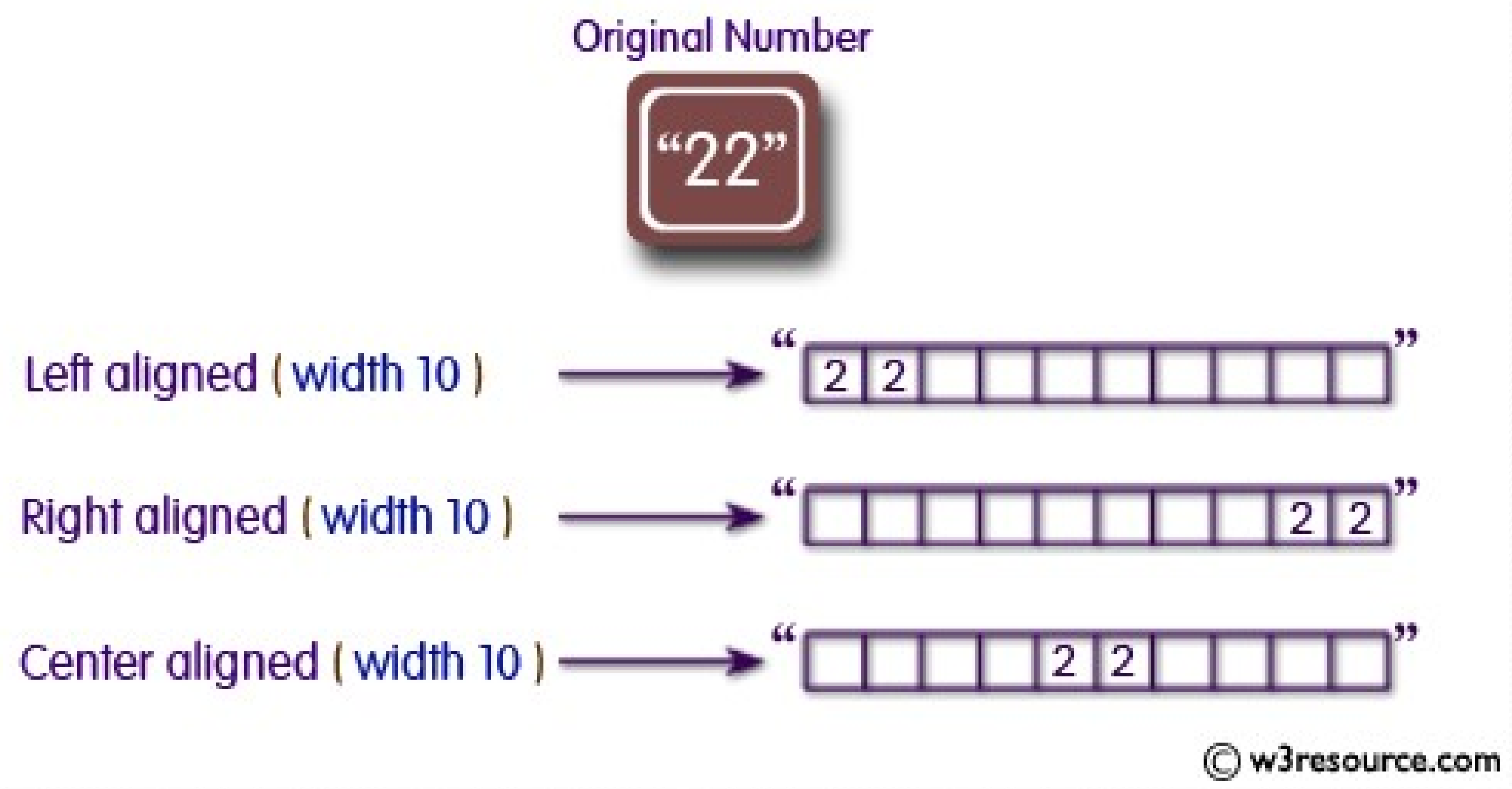
```
Hello Adam, your balance is   230.235
```

We've used the same example from above to show the difference between keyword and positional arguments.

Here, instead of just the parameters, we've used a key-value for the parameters. Namely, name="Adam" and blc=230.2346.

Since, these parameters are referenced by their keys as {name} and {blc:9.3f}, they are known as keyword or named arguments.

Display a number in left, right and center aligned



Original Number
"22"
Left aligned (width 10)
" 2 2 "
Right aligned (width 10)
" 2 2 "
Center aligned (width 10)
" 2 2 "
© w3resource.com

Display a number in left, right and center aligned

```
x = 4500
print("Original Number: ", x)
print("Original Number: ", x)
print("Original Number: ", x)
print("----------------------------------------------------")
print("Left aligned (width 40) {data:<40d}".format(data=x));
print("Right aligned (width 40) {data:>40d}".format(data=x));
print("Center aligned (width 40) {data:^40d}".format(data=x));
print()
```