Python Bootcamp & Masterclass

strings

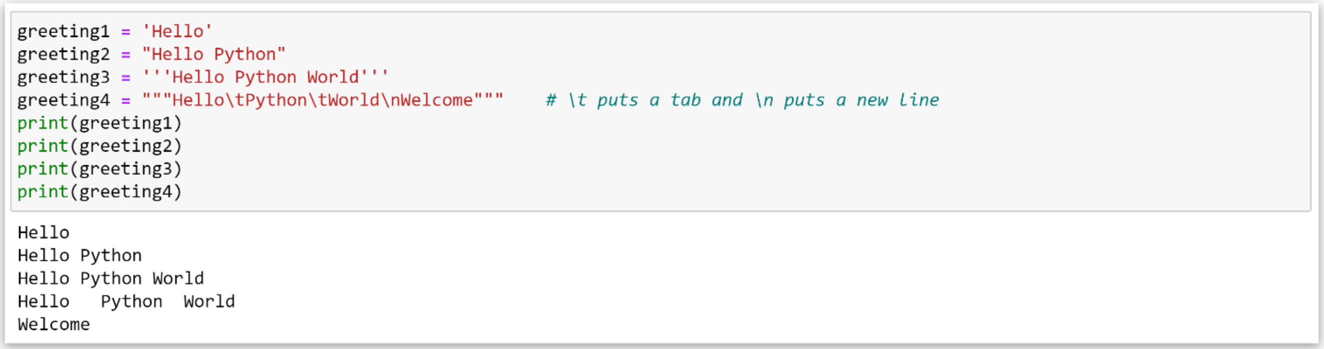
gk nxt

str

Python handles text data with a string object called **str**

A string (**str**) is a sequence of characters and is immutable

Strings should be delimited by single quotes (' ‘) or double quotes (" ") or triple single quotes (''' '''), or triple double quotes (""" """) and can contain tab (\t) and newline (\n) characters.



```python
greeting1 = 'Hello'
greeting2 = "Hello Python"
greeting3 = '''Hello Python World'''
greeting4 = """Hello\tPython\tWorld\nWelcome"""    # \t puts a tab and \n puts a new line
print(greeting1)
print(greeting2)
print(greeting3)
print(greeting4)
```

```
Hello
Hello Python
Hello Python World
Hello	Python	World
Welcome
```

gknxt

String delimiters

The quotes surrounding a string are called delimiters because they tell Python where a string begins and where it ends. When one type of quotes is used as the delimiter, the other type of quote can be used inside of the string

When only one type of quotes need to be used, escape character (\) can be used to tell Python that all the middle quotes are not delimiters

```
india_gdp = "India's GDP is $3.1 trillion in 2021"
india_gdp
```

```
"India's GDP is $3.1 trillion in 2021"
```

```
india_gdp = 'India\'s GDP is $3.1 trillion in 2021'
india_gdp
```

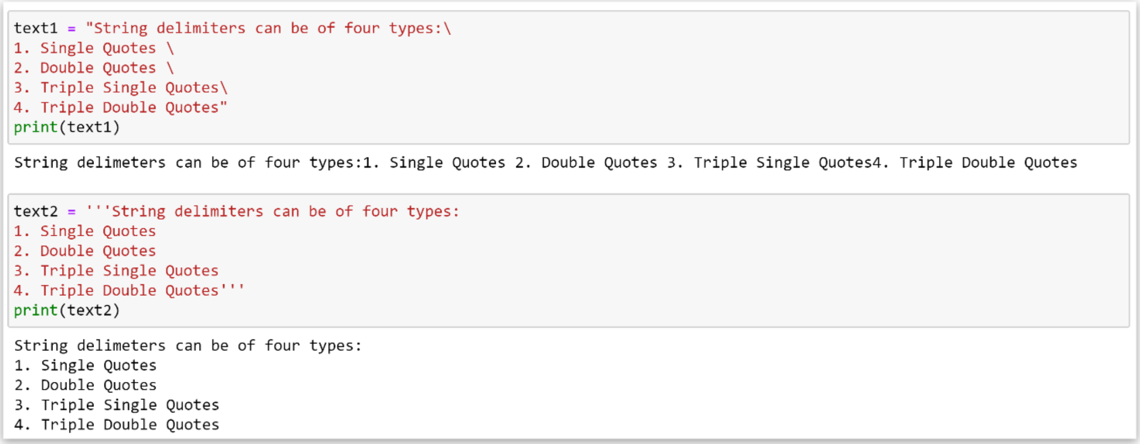
```
"India's GDP is $3.1 trillion in 2021"
```

gknxt

Multiline Strings

The PEP 8 style guide recommends that each line of Python code contain no more than 79 characters - including spaces.

When long strings exceeding 79 characters need to be created, they can be broken into multiple lines by using escape character (\) if formatting is not an issue, or triple quotes if formatting need to be preserved



```python
text1 = "String delimiters can be of four types:\
1. Single Quotes \
2. Double Quotes \
3. Triple Single Quotes\
4. Triple Double Quotes"
print(text1)
```

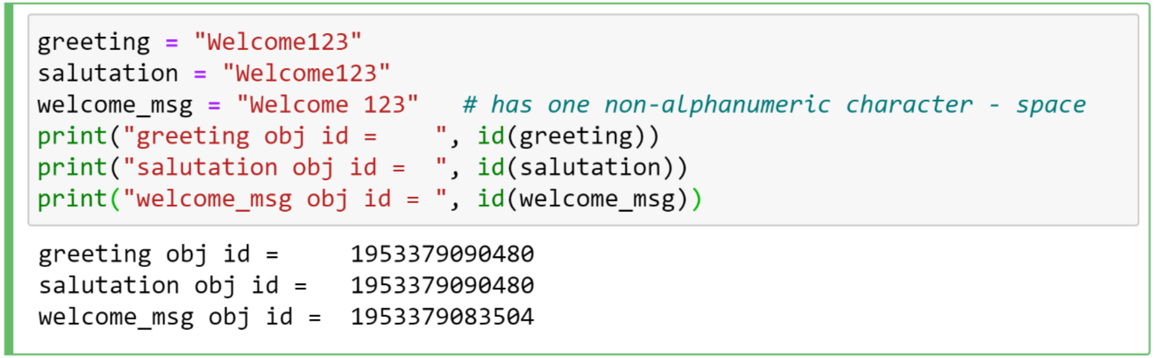
```
String delimeters can be of four types:1. Single Quotes 2. Double Quotes 3. Triple Single Quotes4. Triple Double Quotes
```

```python
text2 = '''String delimiters can be of four types:
1. Single Quotes
2. Double Quotes
3. Triple Single Quotes
4. Triple Double Quotes'''
print(text2)
```

```
String delimeters can be of four types:
1. Single Quotes
2. Double Quotes
3. Triple Single Quotes
4. Triple Double Quotes
```

String Interning

All string objects of exact same character sequence of alphanumeric characters only are shared for execution efficiency. If there are a few string variables whose values are the same, they will be interned by Python implicitly and refer to the same object in the memory to save space and time in string comparison. Starting from Python version 3.7, strings with more than 4096 characters will not be interned as per the AST Optimizer (https://github.com/python/cpython/blob/3.7/Python/ast_opt.c)



```
greeting = "Welcome123"
salutation = "Welcome123"
welcome_msg = "Welcome 123"   # has one non-alphanumeric character - space
print("greeting obj id =    ", id(greeting))
print("salutation obj id =  ", id(salutation))
print("welcome_msg obj id = ", id(welcome_msg))
```

```
greeting obj id =     1953379090480
salutation obj id =   1953379090480
welcome_msg obj id =  1953379083504
```

gknxt

Python has a built-in function that we can use to intern a string explicitly. It is the `intern( )` function in the `sys` module. Explicit interning can be used to overcome 4096 character limit.

```
a = 'K' * 4096                          # https://github.com/python/cpython/blob/3.7/Python/ast_opt.c
b = 'K' * 4096                          # implicit string interning
print("a obj id =    ", id(a))
print("b obj id =    ", id(b))
```

```
a obj id =     1953368198976
b obj id =     1953368198976
```

```
c = 'K' * 4097
d = 'K' * 4097
print("c obj id =    ", id(c))
print("d obj id =    ", id(d))
```

```
c obj id =     1953369707440
d obj id =     1953369407968
```

```
import sys
e = sys.intern('K' * 4097)               # explicit string interning
f = sys.intern('K' * 4097)
print("e obj id =    ", id(e))
print("f obj id =    ", id(f))
```

```
e obj id =     1953369143376
f obj id =     1953369143376
```

String Unpacking

The asterisk (*) operator can be used to unpack iterable objects, and double asterisk (**) operator can be used to unpack dictionaries. Strings are sequences and iterable objects. Unpacking a string breaks the string into the individual characters.

```python
country = 'USA'
print(country)
print(*country)                         #string unpacking with default sep (space)
print(*country, sep='.', end='.')    #string unpacking with default '.' sep
```

```
USA
U S A
U.S.A.
```

String Concatenation

The + operator can be used to explicitly concatenate (join together) string objects.

Python implicitly concatenates two strings if they are just separated by space(s).

```
sci = 'Albert'  ' ' 'Einstein'                    #implicit concatenation
sci
```

```
'Albert Einstein'
```

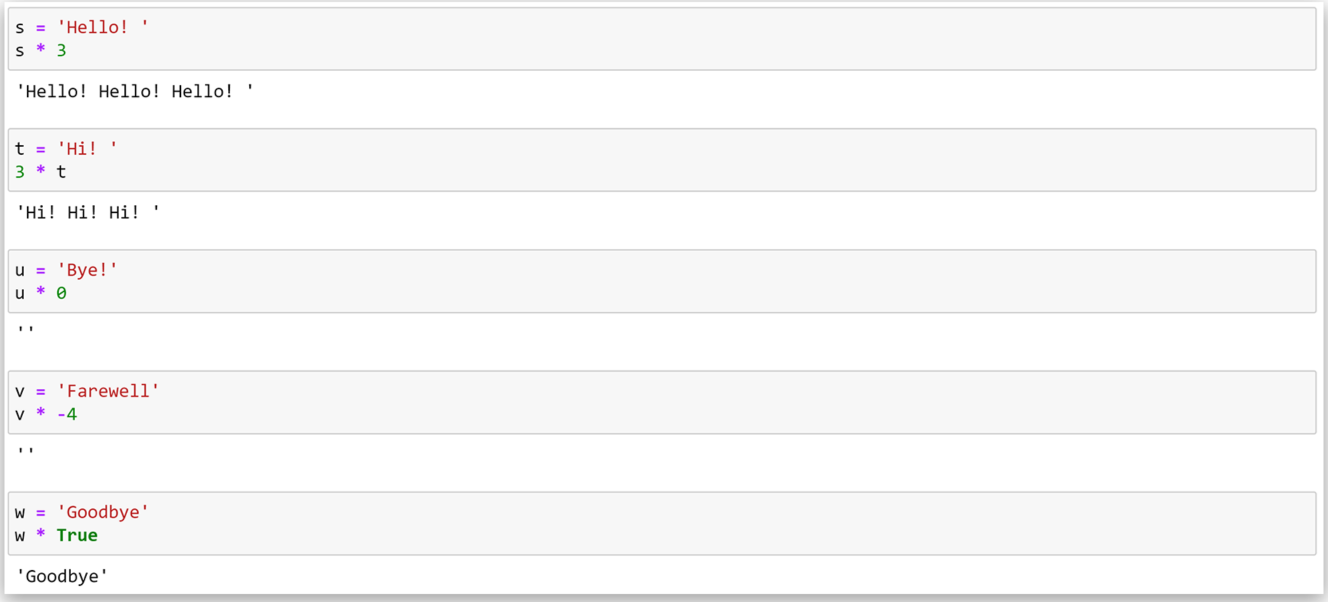
```
mahatma = 'Mohandas' + ' ' + 'K' + ' ' + 'Gandhi'   #explicit concatenation
mahatma
```

```
'Mohandas K Gandhi'
```

gknxt

String Replication

The * operator (asterisk) can be used to replicate (multiply) string objects by using it with a non-empty string and a positive integer.



```
s = 'Hello! '
s * 3
```

```
'Hello! Hello! Hello! '
```

```
t = 'Hi! '
3 * t
```

```
'Hi! Hi! Hi! '
```

```
u = 'Bye!'
u * 0
```

```
''
```

```
v = 'Farewell'
v * -4
```

```
''
```

```
w = 'Goodbye'
w * True
```

```
'Goodbye'
```

gknxt

String indexing

String is a sequence, so it has a numbered position called an index. In Python indexing starts at `0` (largest index in a string is always one less than the string's length)

Each character in a string can be accessed by placing the index in item access operator ([ ])

Strings can be accessed in the reverse order by using negative indices starting with -1 (The last character in a string has index -1, the second-to-last character has index -2, and so on)

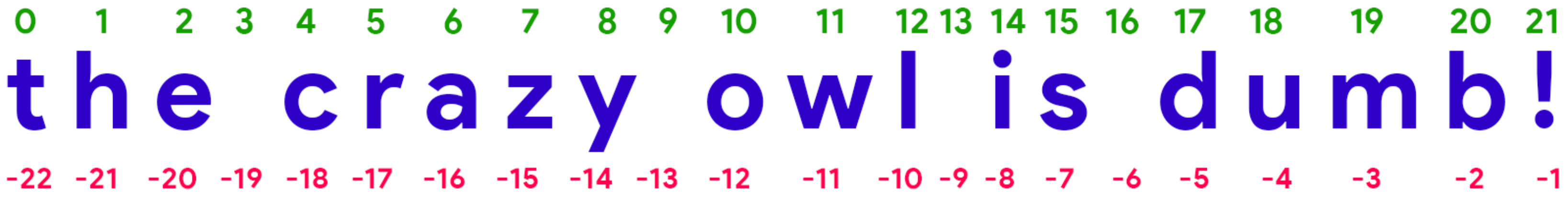
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
t	h	e		c	r	a	z	y		o	w	l		i	s		d	u	m	b	!
-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

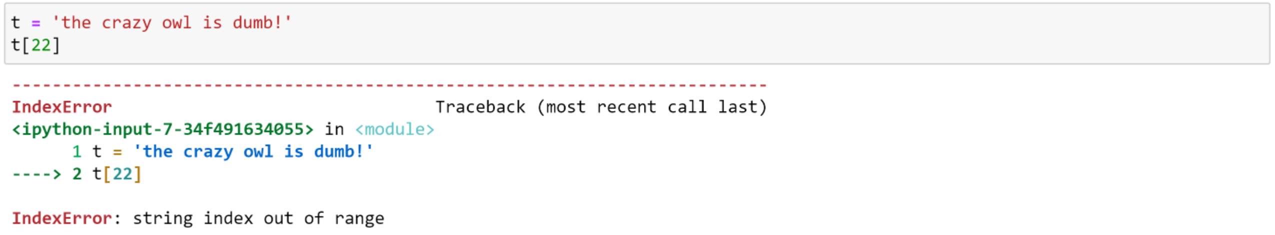
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
t	h	e		c	r	a	z	y		o	w	l		i	s		d	u	m	b	!
-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```python
s = 'the crazy owl is dumb!'
s[0]
s[1]
s[21]
s[-1]
s[-22]
```

```
't'

'h'

'!'

'!'

't'
```

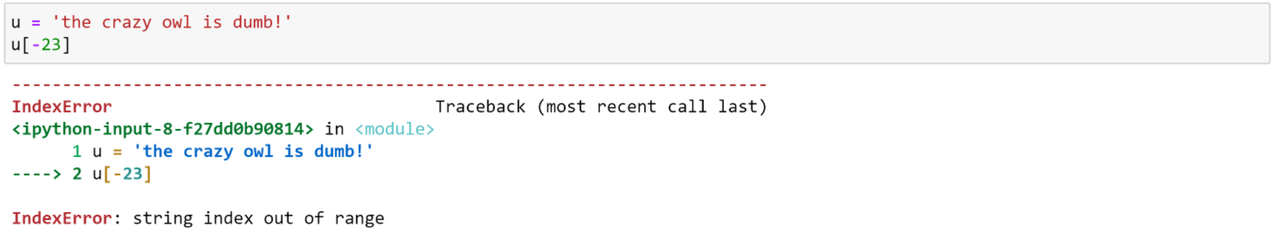
Accessing an index beyond the end of a string or a negative index less than the index of the first character in the string results in an `IndexError`





```
t = 'the crazy owl is dumb!'
t[22]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-7-34f491634055> in <module>
      1 t = 'the crazy owl is dumb!'
----> 2 t[22]

IndexError: string index out of range
```



```
u = 'the crazy owl is dumb!'
u[-23]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-8-f27dd0b90814> in <module>
      1 u = 'the crazy owl is dumb!'
----> 2 u[-23]

IndexError: string index out of range
```

gknxt

String Slicing

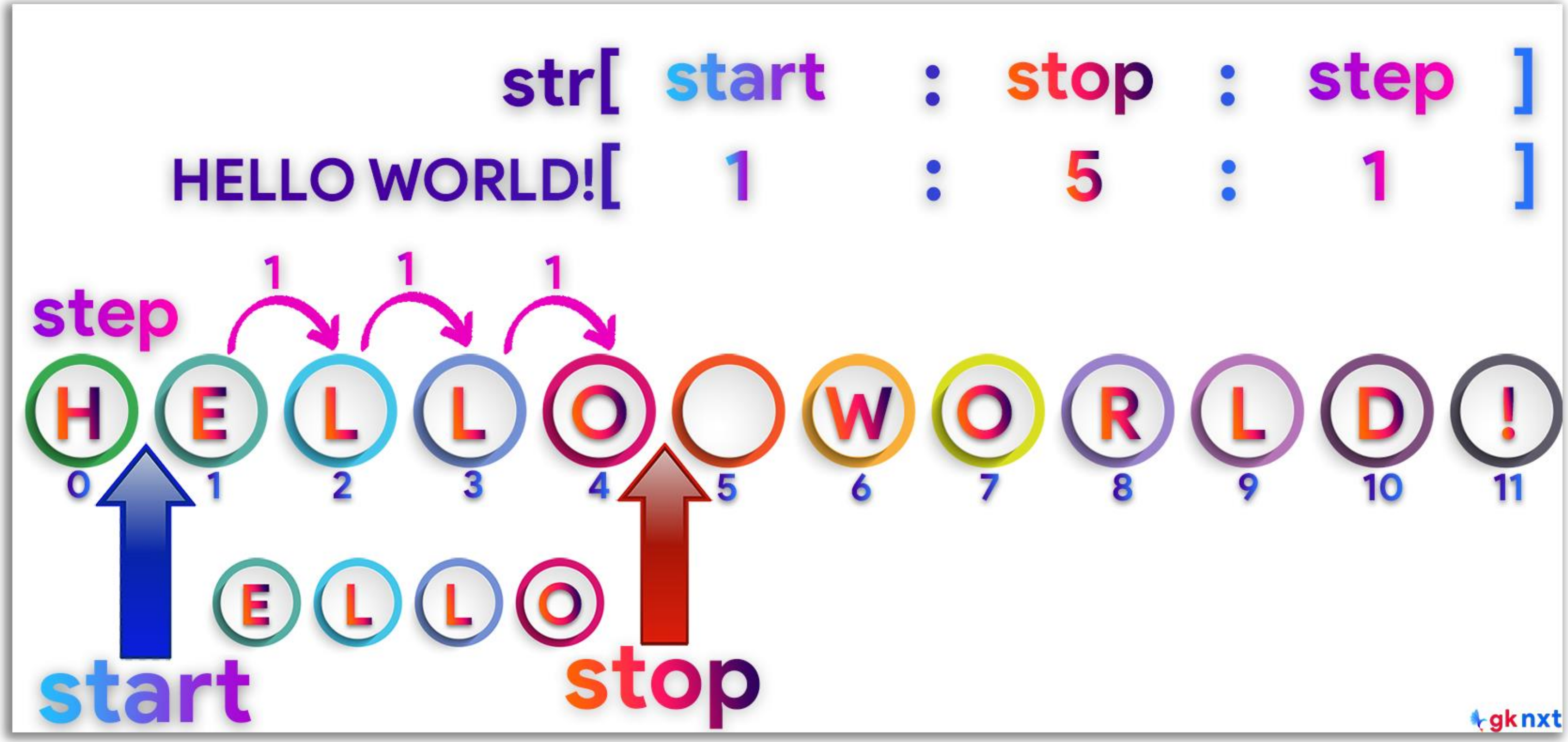
An entire slice (subsequence) of characters can be extracted from a string using the slice operator ([ ]) and colon(s) to separate optional start, optional stop and optional step options.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
t	h	e		c	r	a	z	y		o	w	l		i	s		d	u	m	b	!
-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

seq[start:stop:step]

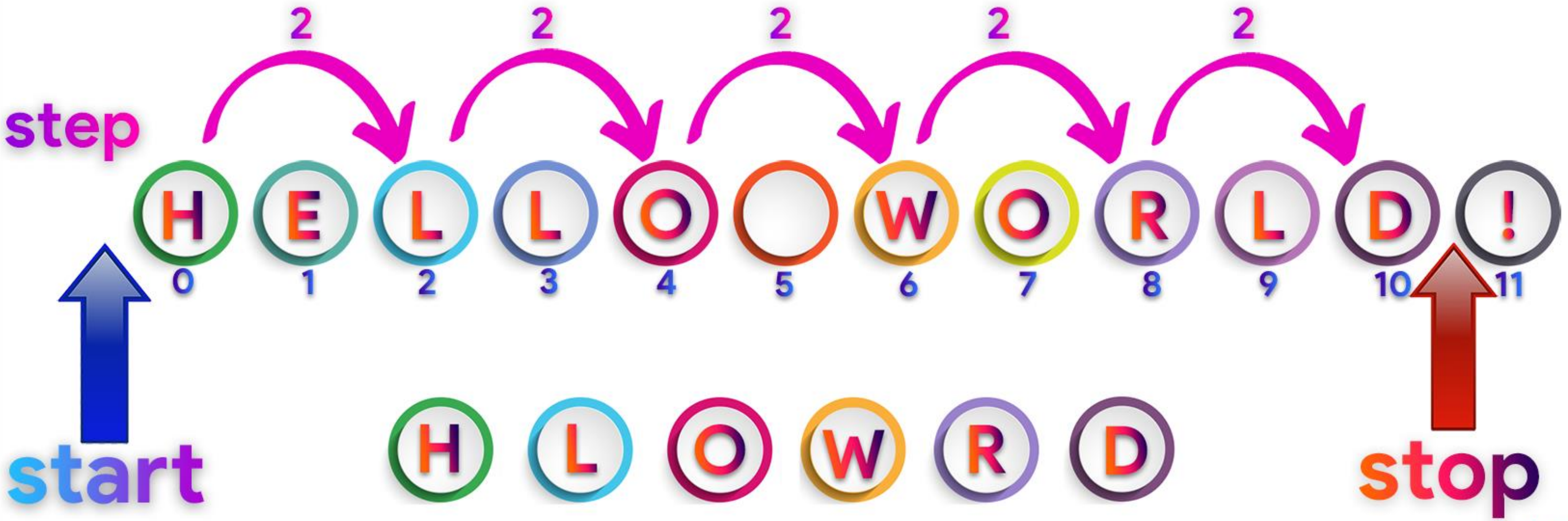
default:	0	len(seq)	1

gknxt


str[start : stop : step]
HELLO WORLD![1 : 5 : 1]
step
1
1
1
H E L L O W O R L D !
0 1 2 3 4 5 6 7 8 9 10 11
E L L O
start
stop
gknxt

gknxt

```
str[ start : stop : step ]
HELLO WORLD![ : 11 : 2 ]
```



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
t	h	e		c	r	a	z	y		o	w	l		i	s		d	u	m	b	!
-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

seq[start:stop:step]

default: 0 len(seq) 1

```
x = 'the crazy owl is dumb!'
x[:]
x[::]
x[0:]
x[:22]
x[::1]
x[0:22]
x[0:22:1]
```

```
'the crazy owl is dumb!'

'the crazy owl is dumb!'

'the crazy owl is dumb!'

'the crazy owl is dumb!'

'the crazy owl is dumb!'

'the crazy owl is dumb!'

'the crazy owl is dumb!'
```

Entire string slicing, from start to end, will just create a new reference, not a new object.

If a part of the string is sliced, not the entire string, a new object will be created

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
t	h	e		c	r	a	z	y		o	w	l		i	s		d	u	m	b	!
-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

seq[	start:	stop:	step]
default:	0	len(seq)	1

```
x = 'the crazy owl is dumb!'
id(x)
id(x[:])
id(x[::])
id(x[0:22:1])
u = x[::]
v = x[0:22:1]
id(u)
id(v)
```

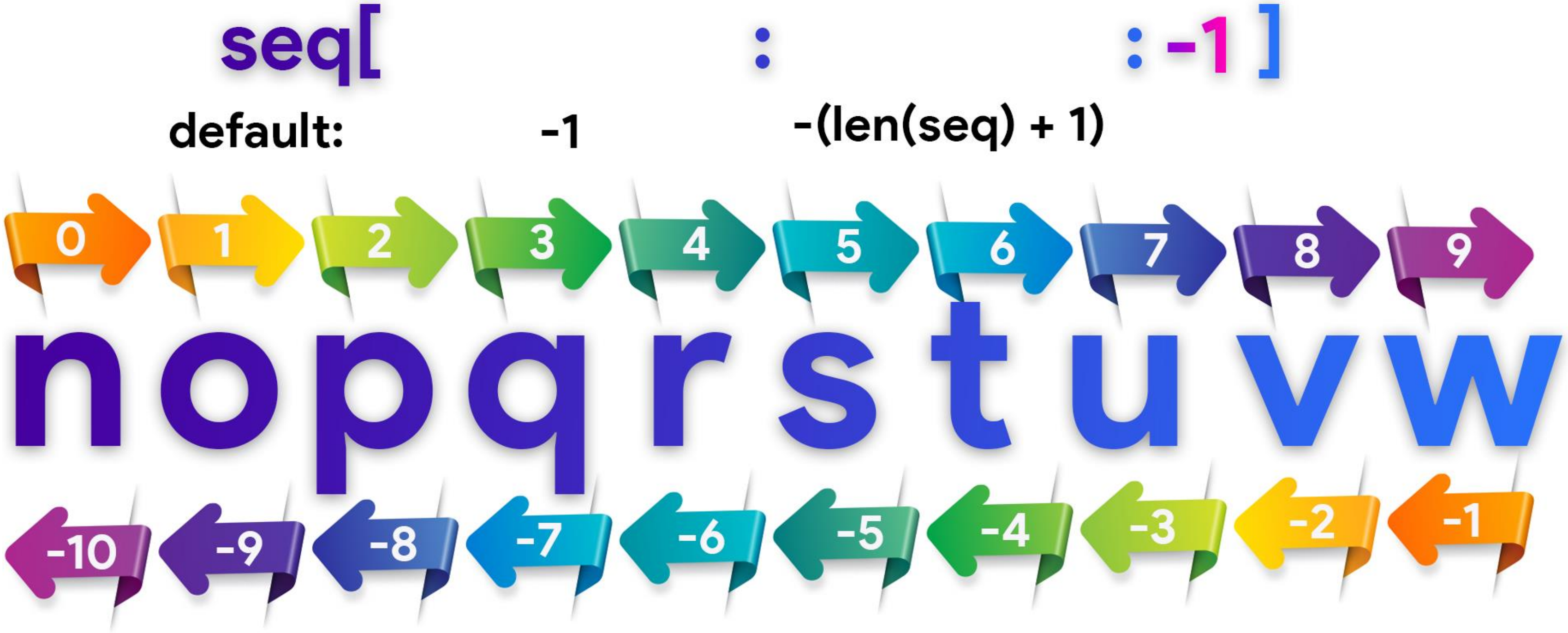
```
1521937613728

1521937613728

1521937613728

1521937613728

1521937613728

1521937613728
```

```
t = 'the crazy owl is dumb!'
u = t[:]
v = t[ : : 2]
w = t[ :20: ]
x = t[1: ]
y = t[ : 100: ]
id(u)
id(v)
id(w)
id(x)
id(y)
```

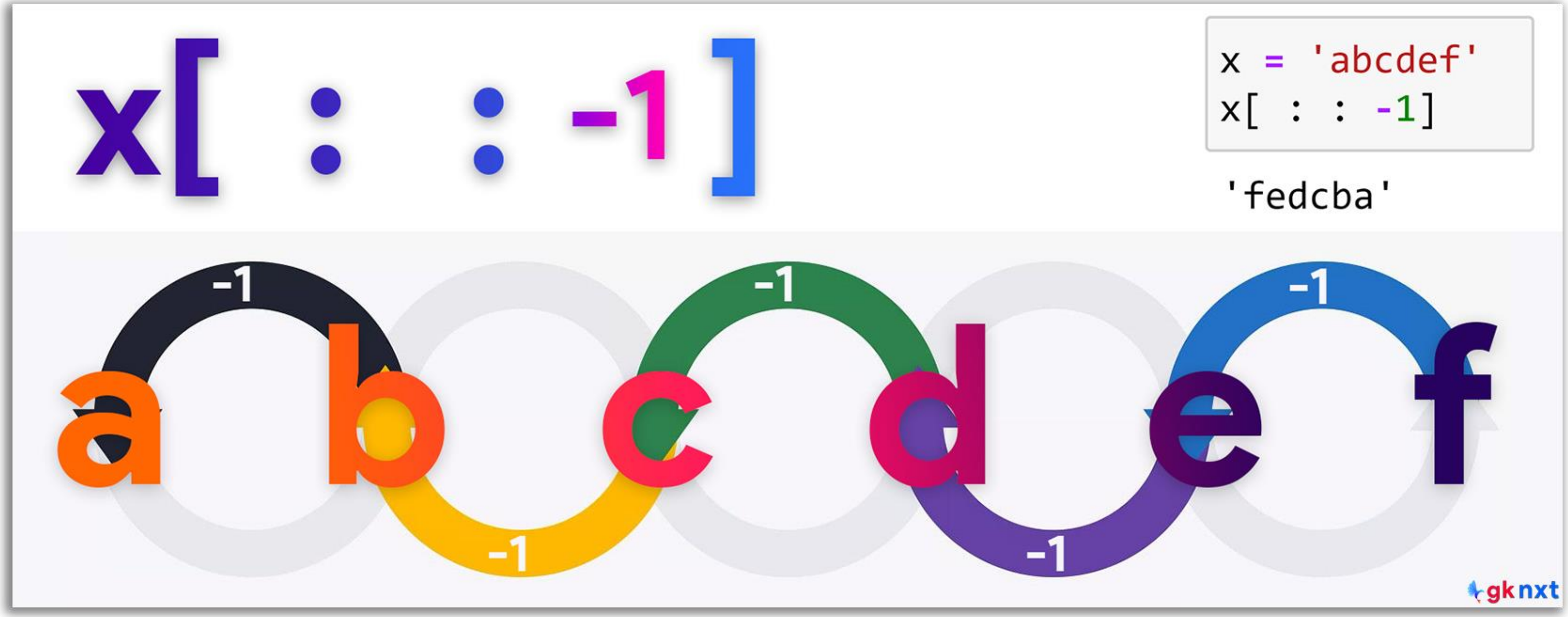
```
2165684511728

2165684589296

2165684561360

2165684512368

2165684511728
```

gknxt

If the step is **negative**, slicing will be in the reverse order and **reverse index** starts at **-1**



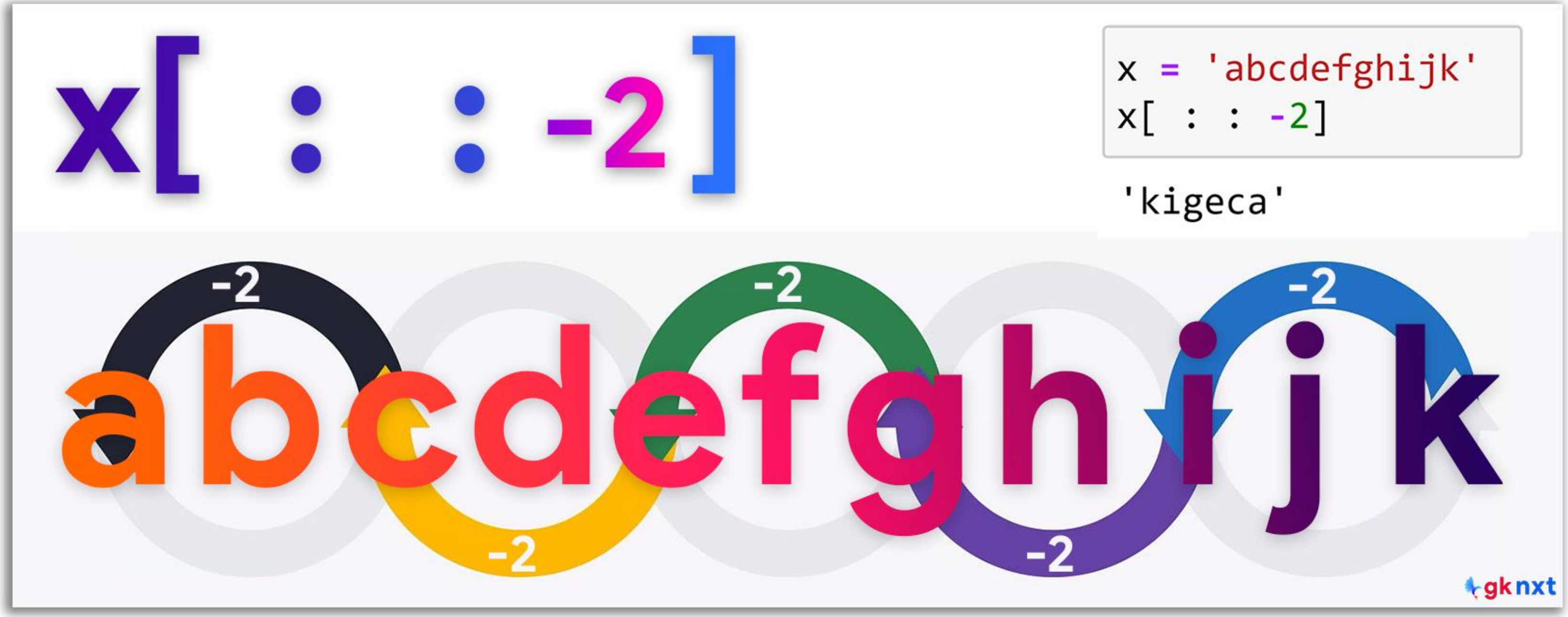
gknxt



```
x = 'abcdef'
x[ : : -1]
```

```
'fedcba'
```

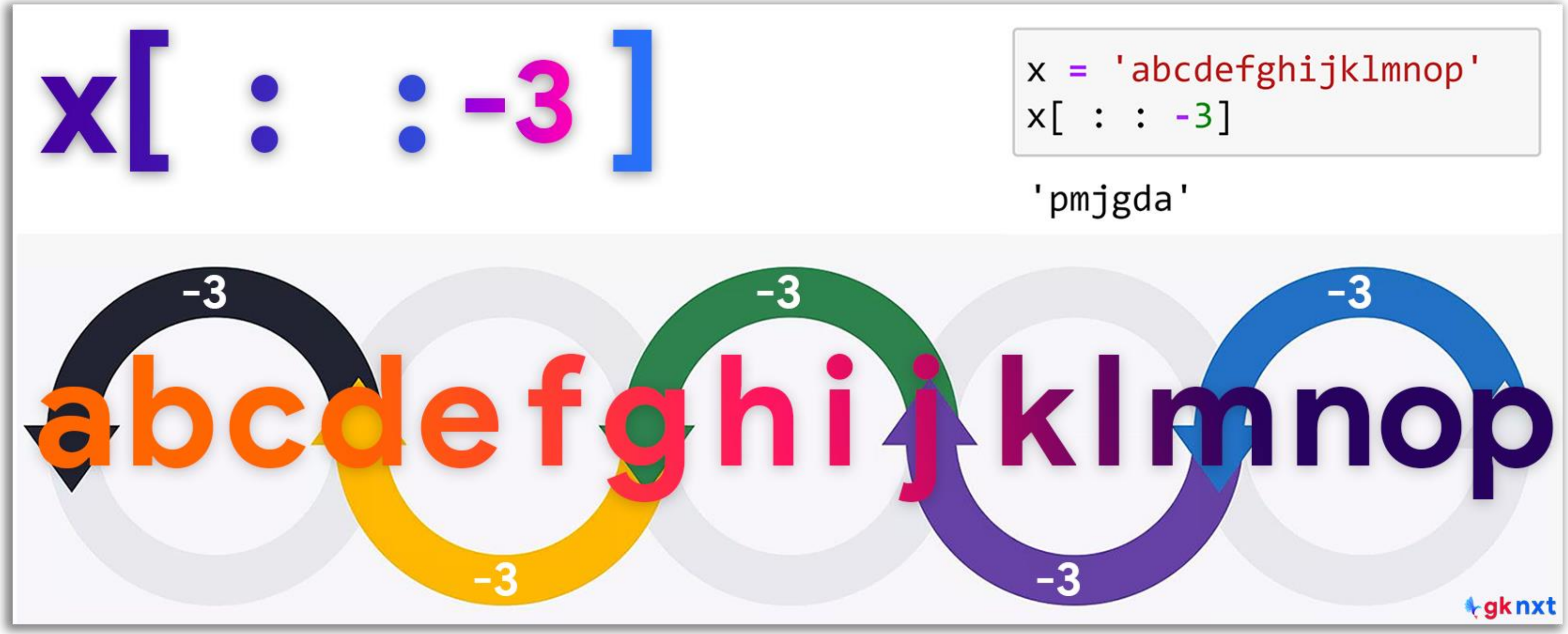
gknxt



```
x = 'abcdefghijk'
x[ : : -2]
```

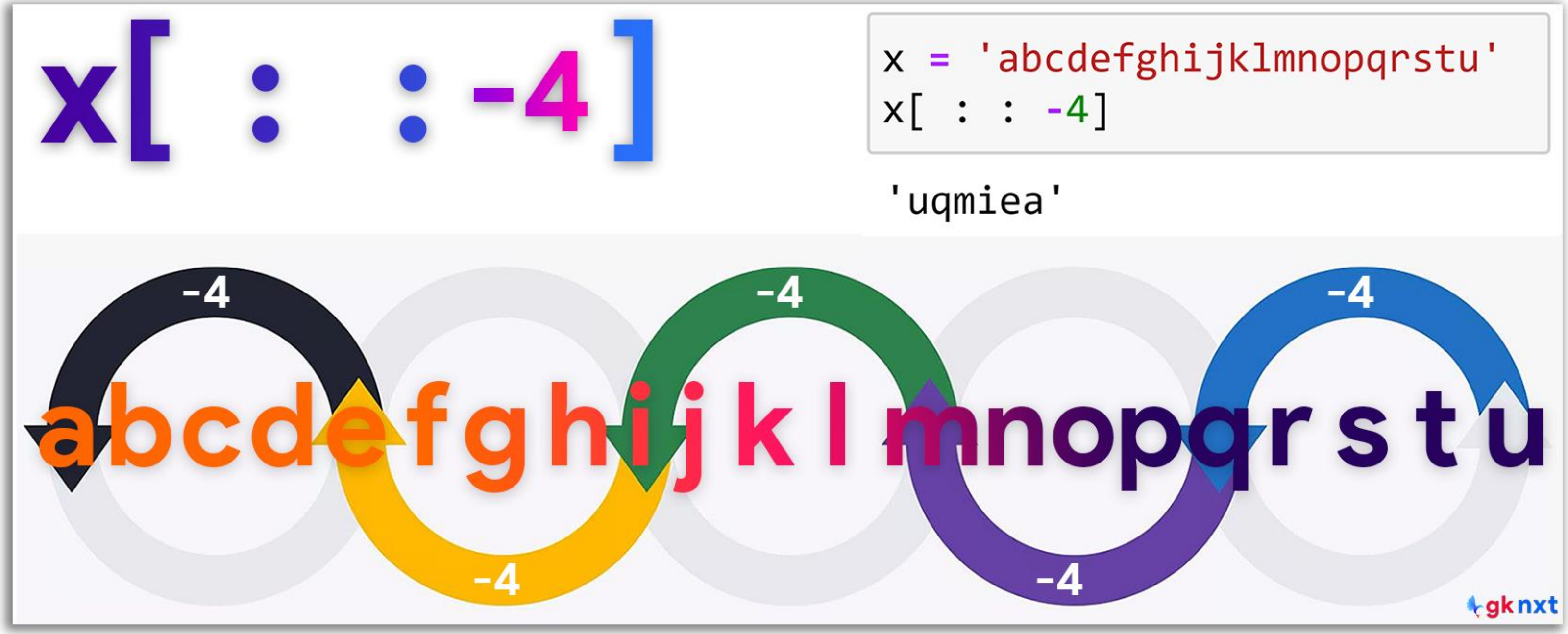
```
'kigeca'
```

gknxt



```
x = 'abcdefghijklmnop'
x[ : : -3]
```

```
'pmjgda'
```



x[: : -4]

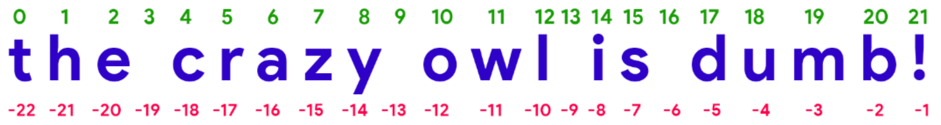
```
x = 'abcdefghijklmnopqrstu'
x[ : : -4]
```

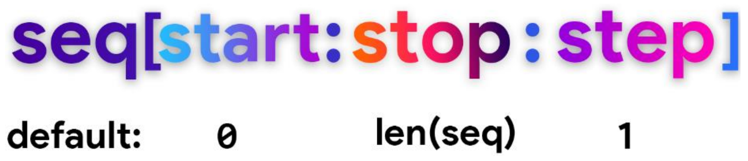
```
'uqmiea'
```

Any or all three indices (start, stop and step) can be beyond the range for the sequence.

If the start is out of range, empty string will return.

If the stop is out of range, it will default to the length of the sequence.





```
x = 'the crazy owl is dumb!'
x[100:]                     # if the start is out of range, empty string will be the slice
x[-100: : -1]               # if the start is out of range for reverse indexing, empty string will be the slice
x[:100]                     # if the stop is out of range, stop defaults to length of sequence
x[: -100: -1]               # if the stop is out of range for reverse indexing, stop defaults to -(length of sequence + 1)
x[: : 100]                  # step can also be out of range
```

```
''

''

'the crazy owl is dumb!'

'!bmud si lwo yzarc eht'

't'
```

Slicing can be chained. A new string object is created when you reverse a string

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
t	h	e		c	r	a	z	y		o	w	l		i	s		d	u	m	b	!
-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

seq[	start:	stop:	step]
default:	0	len(seq)	1

```python
x = 'the crazy owl is dumb!'
r = x[:: -1]
y = x[::-1][::-1]
print(r)
print(x)
print(y)
id(r)
id(x)
id(y)
```

```
!bmud si lwo yzarc eht
the crazy owl is dumb!
the crazy owl is dumb!

1403571343728

1403571346128

1403571402912
```

```python
s = 'the crazy owl is dumb!'
s[::-1][::-5]
s[::-1][::-2]
```

```
'trosb'

'tecayoli ub'
```

immutability

Python strings are immutable (cannot be modified after creation)

```
k = 'Tomatoe'
k[6] = '!'      # as string is immutable, modification is not allowed
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-44f00c783c1c> in <module>
      1 k = 'Tomatoe'
----> 2 k[6] = '!'

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

```
word = "uma"
print(word, id(word))
word = "p" + word[:]
print(word, id(word))
```

```
uma 1403571424176
puma 1403571424752
```

gk nxt

Online Resources

For best python resources, please visit:

gknxt.com/python/

gk nxt

Python Bootcamp & Masterclass

Thank You

for your Rating & Review

gknxt