APT 1025

Introduction to Programming

Sequences: Strings

Sequences

- *Sequences* are collections of data values that are ordered by position
- A *string* is a sequence of characters
- A *list* is a sequence of any Python data values
- A *tuple* is like a list but cannot be modified

Examples

```
a = 'apple'
b = 'banana'
print(a, b)                          # Displays apple banana

fruits = (a, b)                      # A tuple
print(fruits)                        # Displays ('apple', 'banana')

veggies = ['bean', 'lettuce']        # A list
print(veggies)                       # Displays ['bean', 'lettuce']
```

Strings contain characters

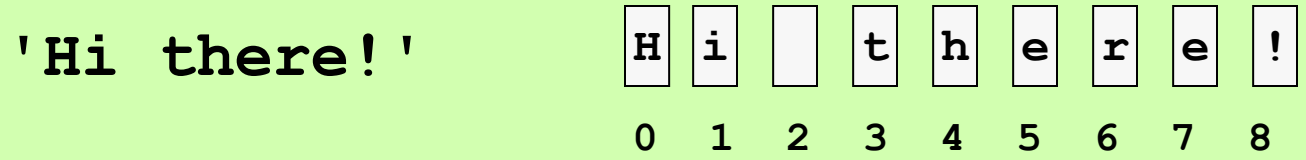
Tuples and lists can contain anything

String Assignment, Concatenation, and Comparisons

```
a = 'apple'
b = 'banana'
print(a + b)          # Displays applebanana
print(a == b)         # Displays False
print(a < b)          # Displays True
```

Strings can be ordered like they are in a dictionary

Positions or Indexes



Each character in a string has a unique position called its *index*

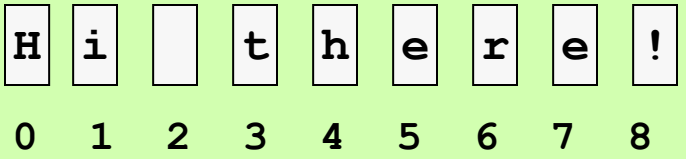
We count indexes from 0 to the length of the string minus 1

A **`for`** loop automatically visits each character in the string, from beginning to end

```
for ch in 'Hi there!': print(ch)
```

Traversing with a **`for`** Loop

`'Hi there!'`



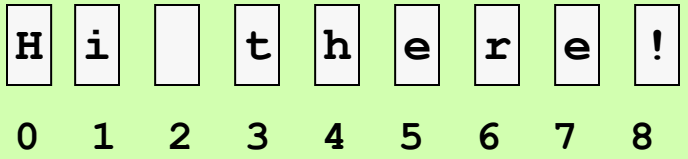
A **`for`** loop automatically visits each character in the string, from beginning to end

```
for ch in 'Hi there!': print(ch, end = '')


# Prints Hi there!
```

Summing with Strings

'Hi there!'



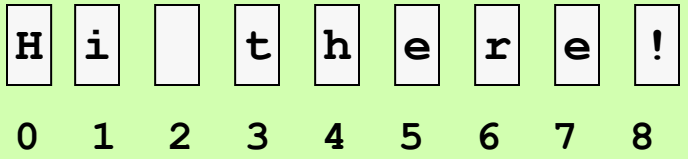
Start with an empty string and add characters to it with **+**

```
noVowels = ''
for ch in 'Hi there!':
    if not ch in ('a', 'e', 'i', 'o', 'u',
                  'A', 'E', 'I', 'O', 'U'):
        noVowels += ch
print(noVowels)

# Prints H thr!
```

The Subscript Operator

`'Hi there!'`



Alternatively, any character can be accessed using the *subscript operator* **`[]`**

This operator expects an **`int`** from 0 to the length of the string minus 1

Example:

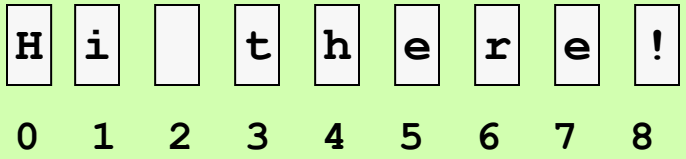
```
'Hi there!'[0]   # equals 'H'
```

Syntax:

```
<a string>[<an int>]
```

The **`len`** Function

`'Hi there!'`



The **`len`** function returns the length of any sequence

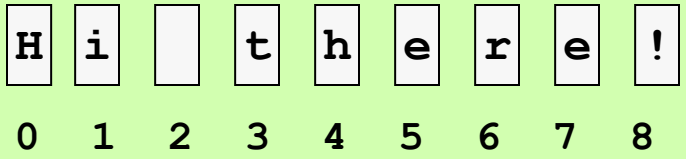
```
>>> len('Hi there!')
9

>>> s = 'Hi there!'

>>> s[len(s) - 1]
'!'
```

An Index-Based Loop

'Hi there!'



If you need the positions during a loop, use the subscript operator

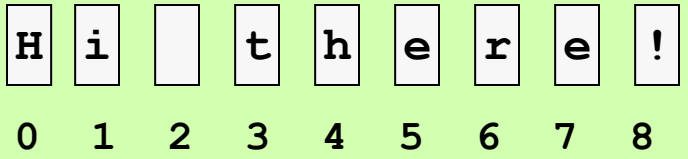
```
s = 'Hi there!'

for ch in s: print(ch)

for i in range(len(s)): print(i, s[i])
```

Oddball Indexes

'Hi there!'



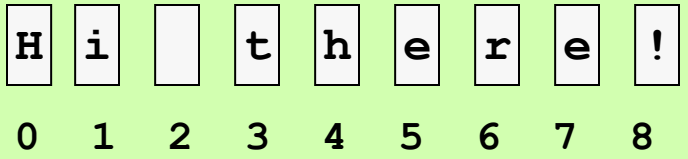
To get to the last character in a string:

```
s = 'Hi there!'

print(s[len(s) - 1])      # Displays !
```

Oddball Indexes

'Hi there!'



To get to the last character in a string:

```
s = 'Hi there!'

print(s[len(s) - 1])

# or, believe it or not,

print(s[-1])
```

A negative index counts backward from the last position in a sequence

Slicing Strings

Extract a portion of a string (a substring)

```
s = 'Hi there!'

print(s[0:])          # Displays Hi there!

print(s[1:])          # Displays i there!

print(s[:2])          # Displays Hi (two characters)

print(s[0:2])         # Displays Hi (two characters)
```

The number to the right of `:` equals one plus the index of the last character in the substring

String Methods

```
s = 'Hi there!'

print(s.find('there'))          # Displays 3

print(s.upper())                # Displays HI THERE!

print(s.replace('e', 'a'))      # Displays Hi thara!

print(s.split())                # Displays ['Hi', 'there!']
```

A *method* is like a function, but the syntax for its use is different:

```
<a string>.<method name>(<any arguments>)
```

String Methods

```
s = 'Hi there!'

print(s.split())                    # Displays ['Hi', 'there!']
```

A sequence of items in **[ ]** is a Python *list*

Getting Help on Strings

```
>>> dir(str)
<a list of all the names defined in the string type>

>>> help(str.split)
<documentation for the string method split>
```

Characters in Computer Memory

- Each character translates to a unique integer called its *ASCII value* (American Standard for Information Interchange)

- Basic ASCII ranges from 0 to 127, for 128 keyboard characters and some control keys

The Basic ASCII Character Set

	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	"	#	$	%	&	`
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	'	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

The **`ord`** and **`chr`** Functions

`ord` converts a single-character string to its ASCII value

`chr` converts an ASCII value to a single-character string

```
print(ord('A'))                    # Displays 65

print(chr(65))                     # Displays A

for ascii in range(128):           # Display 'em all
    print(ascii, chr(ascii))
```

Data Encryption

A really simple (and quite lame) encryption algorithm replaces each character with its ASCII value and a space

```
source = "I won't be here!"

code = ""
for ch in source:
    code = code + str(ord(ch)) + " "
print(code)

# Displays 73 32 119 111 110 39 116 32 98 101 32 104 101 33
```

Data Decryption

To decrypt an encoded message, we split it into a list of substrings and convert these ASCII values to the original characters

```
source = ""
for ascii in code.split():
    source = source + chr(int(ascii))
print(source)          # Displays I won't be here!
```