Working with Strings in Python

Overview of (object) data types in Python

Name	Type	Description
Integers	int	Whole numbers, such as: **3 300 200**
Floating point	float	Numbers with a decimal point: **2.3 4.6 100.0**
Strings	str	Ordered sequence of characters: **"hello" 'Sammy' "2000" "楽しい"**
Lists	list	Ordered sequence of objects: **[10,"hello",200.3]**
Dictionaries	dict	Unordered Key:Value pairs: **{"mykey" : "value" , "name" : "Frankie"}**
Tuples	tup	Ordered immutable sequence of objects: **(10,"hello",200.3)**
Sets	set	Unordered collection of unique objects: **{"a","b"}**
Booleans	bool	Logical value indicating **True** or **False**

Strings

- The next major build-in type is the Python **STRING** --- an **ordered** collection of characters to *store* and *represent* text-based information

- Python strings are categorized as *immutable sequences* --- meaning they have a **left-to-right order** (sequence) and *cannot be changed in place* (immutable)

- This definition of immutable sequences is sufficient for now. You will learn more from examples.

Single- and Double-Quoted

- Single- and Double-Quoted strings **are the same**

>>> ‘Hello World’ , “Hello World”

- The reason for including both is that it allows you to embed a quote character of the other inside a string

>>> “knight’s” , ‘knight”s’

Strings in Action

- **Basic operations:**

1. len() ← length
2. + ← concatenate
3. * ← repetition

len()

- The len build-in function returns the length of strings

```
>>> len('abc')
```


```
>>> a='abc'
>>> len(a)
```

+

- Adding two string objects creates a new string object

```
>>> 'abc' + 'def'
```

```
>>> a='Hello'
>>> b='World'
>>> a + b
>>> a+ ' ' +b
```

Hello World with space

Concatenation of strings

*

- Repetition may seem a bit obscure at first, but it comes in handy in a surprising number of contexts

- For example, to print a line of 80 dashes

```
>>> print '-' * 80
```

Slices and Indexes

Indexes in slices

- Characters in a string are numbered with *indexes* starting at 0:
  - Example:
    `name = "P. Wicks"`

index	0	1	2	3	4	5	6	7
character	P	.		W	i	c	k	s

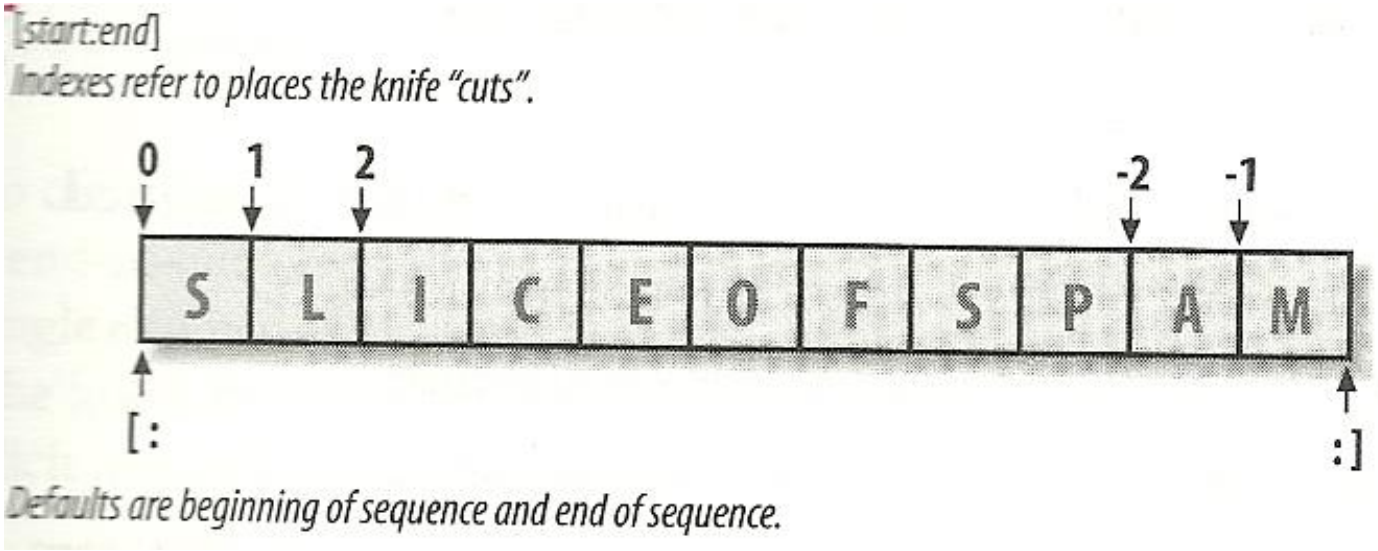
- Accessing an individual character of a string:
  variableName [***index***]

  - Example:
    `print name, "starts with", name[0]`

    Output:
    `P. Wicks starts with P`

More examples on Index and slice

```
>>> aa="SLICEOFSPAM"
>>> aa[0], aa[-2]
        ('S','A')
>>> aa[1:3], aa[1:], aa[:-1]
        ('LI', 'LICEOFSPAM', 'SLICEOFSPA')
```



String Methods: modifying and checking strings assigned to variables

- Assign a string to a variable
- In this case hw = ‘hello world’
- `hw.title()`
- **`hw.upper()`**
- `hw.isdigit()`
- `hw.islower()`

- `dir(str)` lists out all the predefined methods of str

- The string held in your variable remains the same
- The method returns an altered string
- Changing the variable requires reassignment
  - `hw = hw.upper()`
  - `hw` now equals “HELLO WORLD”

Substrings and Methods

```
>>> s = '012345'
>>> s[3]
'3'
>>> s[1:4]
'123'
>>> s[2:]
'2345'
>>> s[:4]
'0123'
>>> s[-2]
'4'
```

- **len**(String) – returns the number of characters in the String

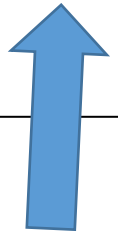
- **str**(Object) – returns a String representation of the Object

```
>>> len(x)
6
>>> str(10.3)
'10.3'
```

String Literals: immutable and overloading

- Strings are *immutable*
- There is no char type like in C++ or Java
- + is overloaded to do concatenation

```
>>> x = 'hello'
>>> x = x + ' there'
>>> x
'hello there'
```



Example

String Literals: Many Kinds

- Can use single or double quotes, and three double quotes for a multi-line string

```
>>> 'I am a string'
'I am a string'
>>> "So am I!"
'So am I!'
>>> s = """And me too!
though I am much longer
than the others :)"""
'And me too!\nthough I am much longer\nthan the others :)'
>>> print s
And me too!
though I am much longer
than the others :)'
```

String Formatting

- Similar to C's printf
- <formatted string> % <elements to insert>
- Can usually just use %s for everything, it will convert the object to its String representation.

```
>>> "One, %d, three" % 2
'One, 2, three'
>>> "%d, two, %s" % (1,3)
'1, two, 3'
>>> "%s two %s" % (1, 'three')
'1 two three'
>>>
```

Characters in strings and limitations on strings

- **string**: A sequence of text characters in a program.
  - Strings start and end with quotation mark `"` or apostrophe `'` characters.
  - Examples:

```
"hello"
"This is a string"
"This, too, is a string.   It can be very long!"
```

- A string may not span across multiple lines or contain a " character.

```
"This is not
a legal String."
"This is not a "legal" String either."
```

- A string can represent characters by preceding them with a backslash.
  - `\t` tab character
  - **`\n` new line character**
  - `\"` quotation mark character
  - `\\` backslash character
  - Example: `"Hello\tthere\nHow are you?"`

Examples of Strings

```
• "hello"+"world"   "helloworld"        # concatenation
• "hello"*3         "hellohellohello"   # repetition
• "hello"[0]        "h"                 # indexing
• "hello"[-1]       "o"                 # (from end)
• "hello"[1:4]      "ell"               # slicing
• "hello"[::2]      "hlo"           # slicing with step 2
• len("hello")      5                   # size

• "hello" < "jello"     1               # comparison
• "e" in "hello"        1               # search

• New line:             "escapes: \n "
• Line continuation:    triple quotes '''
• Quotes:           'single quotes', "raw strings"
```