



## Sequence of characters

- We've talked about strings being a sequence of characters.
- A string is indicated between ' ' or " "
- The exact sequence is maintained



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

2

## And then there is "" "" ""

- triple quotes preserve both the vertical and horizontal formatting of the string
- allows you to type tables, paragraphs, whatever and preserve the formatting

```
""" this is
a test
today"""
```



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

3

## Strings

Can use single or double quotes:

- S = "spam"
- s = 'spam'

Just don't mix them

- myStr = 'hi mom' → ERROR

Inserting an apostrophe:

- A = "knight's" # *mix up the quotes*
- B = 'knight\'s' # *escape single quote*



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

4

## Basic String Operations

```
s = 'spam'
■ length operator len()
len(s)
4
■ + is concatenate
newStr = 'spam' + '-' + 'spam-'
print newStr
spam-spam-
■ * is repeat, the number is how many times
newStr * 3
spam-spam-spam-spam-spam-
```



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

5

## Membership operations

- can check to see if a substring exists in the string, the `in` operator. Returns True or False

```
myStr = 'aabbccdd'
'a' in myStr ⇒ True
'abb' in myStr ⇒ True
'x' in myStr ⇒ False
```



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

6

## Indexing Strings

character	h	e	l	l	o		w	o	r	l	d	
index	0	1	2	3	4	5	6	7	8	9	10	
										...	-2	-1

- every character has an index, starting at 0
- the index operator is []

```
myStr = 'hello world'
myStr[2] => 'l'
myStr[-1] => 'd'
myStr[11] => ERROR, index out of range
```

Michigan State University  
CSE 231, Spring 2009

Intro to Strings

7

## Slicing, the rules

- slicing is the ability to select a subsequence of the overall sequence
- uses the syntax [start : finish], where:
  - start is the index of where we start the subsequence
  - finish is the index of **one after** where we end the subsequence
- if either start or finish are not provided, it defaults to the beginning of the sequence for start and the end of the sequence for finish

Michigan State University  
CSE 231, Spring 2009

Intro to Strings

8

## half open range for slices

- slicing uses what is called a half-open range
- the first index is included in the sequence
- the last index is one **after** what is included

Michigan State University  
CSE 231, Spring 2009

Intro to Strings

9

## Example

char	h	e	l	l	o		w	o	r	l	d
index	0	1	2	3	4	5	6	7	8	9	10
							↑				↑
							first				last

```
myStr = 'hello world'
myStr[6:10] => 'worl'
```

- half-open range means that 'd' is not included. The last index is one after the character included in the slice

Michigan State University  
CSE 231, Spring 2009

Intro to Strings

10

## Examples, defaults

char	h	e	l	l	o		w	o	r	l	d
index	0	1	2	3	4	5	6	7	8	9	10
							↑				↑
							first				last

- myStr[6:] => 'world'

char	h	e	l	l	o		w	o	r	l	d
index	0	1	2	3	4	5	6	7	8	9	10
	↑						↑				
	first						last				

```
myStr[:8] => 'hello wo'
```

Michigan State University  
CSE 231, Spring 2009

Intro to Strings

11

## Count by argument

- also takes three arguments:
  - [start:finish:countBy]
- defaults are:
  - start is beginning, finish is end, countBy is 1

```
myStr = 'hello world'
myStr[0:11:2] => 'hlowrd'
```

- every other letter

Michigan State University  
CSE 231, Spring 2009

Intro to Strings

12

## Some python “idioms”

- idioms are python “phrases” that are used for a common task that might be less obvious to non-python folk
- how to make a copy of a string:
 

```
myStr = 'hi mom'
newStr = myStr[:]
```
- how to reverse a string
 

```
myStr = 'madam I'm adam'
reverseStr = myStr[::-1]
```



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

13

## Indexing S[i] Overview

- S[0] fetches the first item
- Negative indices count backwards from end
- S[-2] fetches the second from the end



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

14

## Slicing S[i:] Overview

- Bounds are between elements.
- Upper bound is not inclusive.
- Defaults are 0 and end, if omitted
- S[1:4] fetches from 1 up to, but not including 4
- S[1:] fetches from 1 to end
- S[:4] fetches from beginning up to, but not including 4
- S[:-1] fetches from beginning up to, but not including the last item
- S[:] fetches all  
(A *top-level* copy of S – more later.)



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

15

## Strings are immutable

- strings are immutable, that is you cannot change one once you make it
- However, you can use it to make another string (copy it, slice it, etc.)
- For example, the reverse string idiom doesn't change the existing string, it simply makes a new string with the letters reversed!



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

16

## Changing strings

- can't change a string in place
 

```
aStr = 'spam'
aStr[1] = 'l' ⇒ ERROR
```
- you can however make new strings
 

```
aStr = aStr[:1] + 'l' + aStr[2:]
print x ⇒ slam
```
- make a new string, reassign it
 

```
aStr = aStr.replace('l', 'c')
print x ⇒ scam
```



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

17

## Replace

- S.replace(old, new, max)
- s = 'hello'
- s = s.replace('l', 'x', 1)
- print s
- hexlo



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

18

## Method

- Methods operate on an object.
- For example, `s.replace('l', 'x', 1)` does a replace on string `s` (a.k.a. object `s`) by replacing 'l' with 'x' once (1).
- There are many string methods which perform operations on string objects.

Let's look at some...



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

19

## Find

- `x = 'hello'`
- `x.find('l')` # find index of 'l' in `x`
- 2

Note how the method 'find' operates on the string object 'x' and the two are associated by using the "dot" notation: `x.find('l')`.

Terminology: the thing(s) in parenthesis, i.e. the 'l' in this case, is called an **argument**.



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

20

## Split

(possibly the most useful string method)

```
s = 'Always look on the bright side of life.'
y = s.split()
print y
  □ ['Always', 'look', 'on', 'the', 'bright', 'side', 'of', 'life.']
print y[1]
  □ look
```

Note the "dot" notation: `s.split()` and the absence of any "arguments"



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

21

## More methods

(even more exist: <http://docs.python.org/lib/string-methods.html>)

- `s.capitalize`
- `s.center(width)`
- `s.count(sub, [start, end])`
- `s.ljust(width)`
- `s.lower()`
- `s.upper()`
- `s.lstrip()`
- `s.find(sub, [start, end])`
- `s.splitlines([keepends])`
- `s.strip()`
- `s.translate(table [, delchars])`



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

22

## String Extras

- Raw Strings so you can include "\"
- ```
x = r'C:\Python25\'
```



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

23

Palindrome  
Examples

## String comparisons, single char

- There are two systems for representing characters: ASCII and Unicode
- ASCII takes the English letters, numbers and punctuation marks and associates them with an integer number
- Single character comparisons are based on that number

Michigan State University CSE 231, Spring 2009 Intro to Strings 25

| Dec | Hex | Oct | Char                     | Dec | Hex | Oct | Char                  | Dec | Hex | Oct | Char   | Dec | Hex | Oct | Char    |
|-----|-----|-----|--------------------------|-----|-----|-----|-----------------------|-----|-----|-----|--------|-----|-----|-----|---------|
| 0   | 000 | 000 | (null)                   | 32  | 20  | 040 | #32: Space            | 64  | 40  | 100 | #64: @ | 96  | 60  | 140 | #96: P  |
| 1   | 001 | 001 | (start of heading)       | 33  | 21  | 041 | #33: ' ' (space)      | 65  | 41  | 101 | #65: A | 97  | 61  | 141 | #97: Q  |
| 2   | 002 | 002 | (start of text)          | 34  | 22  | 042 | #34: "                | 66  | 42  | 102 | #66: B | 98  | 62  | 142 | #98: R  |
| 3   | 003 | 003 | (end of text)            | 35  | 23  | 043 | #35: #                | 67  | 43  | 103 | #67: C | 99  | 63  | 143 | #99: S  |
| 4   | 004 | 004 | (end of transmission)    | 36  | 24  | 044 | #36: \$               | 68  | 44  | 104 | #68: D | 100 | 64  | 144 | #100: T |
| 5   | 005 | 005 | (enquiry)                | 37  | 25  | 045 | #37: %                | 69  | 45  | 105 | #69: E | 101 | 65  | 145 | #101: U |
| 6   | 006 | 006 | (acknowledge)            | 38  | 26  | 046 | #38: &                | 70  | 46  | 106 | #70: F | 102 | 66  | 146 | #102: V |
| 7   | 007 | 007 | (bell)                   | 39  | 27  | 047 | #39: ' (single quote) | 71  | 47  | 107 | #71: G | 103 | 67  | 147 | #103: W |
| 8   | 010 | 010 | (backspace)              | 40  | 28  | 050 | #40: (                | 72  | 48  | 110 | #72: H | 104 | 68  | 150 | #104: X |
| 9   | 011 | 011 | (horizontal tab)         | 41  | 29  | 051 | #41: )                | 73  | 49  | 111 | #73: I | 105 | 69  | 151 | #105: Y |
| 10  | 012 | 012 | (NL line feed, new line) | 42  | 2A  | 052 | #42: *                | 74  | 4A  | 112 | #74: J | 106 | 6A  | 152 | #106: Z |
| 11  | 013 | 013 | (vertical tab)           | 43  | 2B  | 053 | #43: +                | 75  | 4B  | 113 | #75: K | 107 | 6B  | 153 | #107: [ |
| 12  | 014 | 014 | (FF form feed, new page) | 44  | 2C  | 054 | #44: ,                | 76  | 4C  | 114 | #76: L | 108 | 6C  | 154 | #108: \ |
| 13  | 015 | 015 | (carriage return)        | 45  | 2D  | 055 | #45: -                | 77  | 4D  | 115 | #77: M | 109 | 6D  | 155 | #109: ] |
| 14  | 016 | 016 | (shift out)              | 46  | 2E  | 056 | #46: .                | 78  | 4E  | 116 | #78: N | 110 | 6E  | 156 | #110: ^ |
| 15  | 017 | 017 | (shift in)               | 47  | 2F  | 057 | #47: /                | 79  | 4F  | 117 | #79: O | 111 | 6F  | 157 | #111: _ |
| 16  | 020 | 020 | (data link escape)       | 48  | 30  | 060 | #48: 0                | 80  | 50  | 120 | #80: P | 112 | 70  | 160 | #112: ` |
| 17  | 021 | 021 | (device control 1)       | 49  | 31  | 061 | #49: 1                | 81  | 51  | 121 | #81: Q | 113 | 71  | 161 | #113: a |
| 18  | 022 | 022 | (device control 2)       | 50  | 32  | 062 | #50: 2                | 82  | 52  | 122 | #82: R | 114 | 72  | 162 | #114: b |
| 19  | 023 | 023 | (device control 3)       | 51  | 33  | 063 | #51: 3                | 83  | 53  | 123 | #83: S | 115 | 73  | 163 | #115: c |
| 20  | 024 | 024 | (device control 4)       | 52  | 34  | 064 | #52: 4                | 84  | 54  | 124 | #84: T | 116 | 74  | 164 | #116: d |
| 21  | 025 | 025 | (negative acknowledge)   | 53  | 35  | 065 | #53: 5                | 85  | 55  | 125 | #85: U | 117 | 75  | 165 | #117: e |
| 22  | 026 | 026 | (synchronous idle)       | 54  | 36  | 066 | #54: 6                | 86  | 56  | 126 | #86: V | 118 | 76  | 166 | #118: f |
| 23  | 027 | 027 | (end of trans. block)    | 55  | 37  | 067 | #55: 7                | 87  | 57  | 127 | #87: W | 119 | 77  | 167 | #119: g |
| 24  | 030 | 030 | (cancel)                 | 56  | 38  | 070 | #56: 8                | 88  | 58  | 130 | #88: X | 120 | 78  | 170 | #120: h |
| 25  | 031 | 031 | (end of medium)          | 57  | 39  | 071 | #57: 9                | 89  | 59  | 131 | #89: Y | 121 | 79  | 171 | #121: i |
| 26  | 032 | 032 | (substitute)             | 58  | 3A  | 072 | #58: :                | 90  | 5A  | 132 | #90: Z | 122 | 7A  | 172 | #122: j |
| 27  | 033 | 033 | (escape)                 | 59  | 3B  | 073 | #59: ;                | 91  | 5B  | 133 | #91: [ | 123 | 7B  | 173 | #123: k |
| 28  | 034 | 034 | (file separator)         | 60  | 3C  | 074 | #60: <                | 92  | 5C  | 134 | #92: \ | 124 | 7C  | 174 | #124: l |
| 29  | 035 | 035 | (group separator)        | 61  | 3D  | 075 | #61: =                | 93  | 5D  | 135 | #93: ] | 125 | 7D  | 175 | #125: m |
| 30  | 036 | 036 | (record separator)       | 62  | 3E  | 076 | #62: >                | 94  | 5E  | 136 | #94: ^ | 126 | 7E  | 176 | #126: n |
| 31  | 037 | 037 | (unit separator)         | 63  | 3F  | 077 | #63: ?                | 95  | 5F  | 137 | #95: _ | 127 | 7F  | 177 | #127: o |

Michigan State University CSE 231, Spring 2009 Intro to Strings 26 Source: www.asciiabn.com

## comparisons within sequence

- It makes sense to compare within a sequence (lower case, upper case, digits).
  - 'a' < 'b' True
  - 'A' < 'B' True
  - '1' < '9' True
- Can be weird outside of the sequence
  - 'a' < 'A' False
  - 'a' < '0' False

Michigan State University CSE 231, Spring 2009 Intro to Strings 27

## Whole strings

- Compare the first element of each string
  - if they are equal, move on to the next character in each
  - if they are not equal, the relationship between those to characters are the relationship between the string
  - if one ends up being shorter (but equal), the shorter is smaller

Michigan State University CSE 231, Spring 2009 Intro to Strings 28

## examples

- 'a' < 'b' True
- 'aab' < 'aac'
  - first difference is at the last char. 'b' < 'c' so 'aab' is less than 'aac'. True
- 'aa' < 'aaz'
  - The first string is the same but shorter. Thus it is "smaller". True

Michigan State University CSE 231, Spring 2009 Intro to Strings 29

iteration

## iteration through a sequence

- To date we have seen the while loop as a way to iterate over a suite (a group of python statements)
- There is a separate, and very powerful approach available to iterate over all the elements of a sequence, such as the elements of a list or a string



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

31

## for statement

We use the for statement to process each element of a list, one element at a time

```
for item in sequence:
    suite
```



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

32

## What for means

```
myStr='abc'
for myVar in 'abc':
    print myVar
```

- first time through, myVar='a' (myStr[0])
- second time through, myVar='b' (myStr[1])
- third time through, myVar='c' (myStr[2])
- no more sequence left, we quit



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

33

## Power of the for statement

- Sequence iteration as provided by the for state is very powerful and very useful in python.
- Allows you to write some very “short” programs that do powerful things.



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

34

## String formatting

CSE 231, Bill Punch

## String formatting, better printing

- So far, we have just used the defaults of the print function
- We can do many more complicated things to make that output “prettier” and more pleasing.
- We will apply it to our “display” function



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

36

## Basic form

- To understand string formatting, it is probably better to start with an example.

```
print "%s is %d years old"%( "Bill",25)

prints Bill is 25 years old
```



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

37

## More detail

```
print "%s is %d years old" % ("Bill",25)
```

string to print that contains format information (preceeded by %)

% separates the two

object tuple that matches objects to format elements (1 to 1)



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

38

## Format string

- The format string contains a set of format descriptors that describe how an object is to be printed
- Overall:  
%[name][flags][width][.precision]code

where [ ] are optional



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

39

## Many descriptors

- %s string
- %d decimal
- %e floating point exponent
- %f floating point decimal
- %u unsigned integer
- and others



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

40

## matching object to descriptor

- Objects are matched in order with format descriptors. The substitution is made and resulting string printed

```
print "%s is %d years old" % ("Bill",25)

prints Bill is 25 years old
```



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

41

## Width

```
print "%10s is %-10d years old" % ("Bill",25)
```

string 10 spaces wide including the object right justified

decimal 10 spaces wide including the object "-" means left justified

```
prints:      Bill is 25 years old
           {10 spaces} {10 spaces}
           this space in the format string already
```



Michigan State University  
CSE 231, Spring 2009

Intro to Strings

42

## Precision

- `print math.pi`
  - 3.14159265359
- `print "%.4f" % math.pi`
  - 3.1416 (4 decimal points of precision, with rounding)
- `print "%10.2f" % math.pi`
  - 3.14 (10 spaces total including the number and the decimal point)

