7. String Methods

Topics:

Using Methods from the string class

Iterating through a string with `for`

Data + Functions Together

"The square root of nine is three."

The tone of this comment is that the square root function can be applied to numbers like nine.

"Three is nine's square root."

The tone of this comment is that the number nine (like all numbers) comes equipped with a sqrt function.

A new point of view

Methods

A special kind of function that is very important to object-oriented programming is called a method.

In this style of programming, there is a tight coupling between structured data and the methods that work with that data.

Methods

Hard to appreciate the reasons for this coupling between data and methods so early in the course.

For now, we settle on getting used to the special notation that is associated with the use of methods.

We will get into this topic using strings.

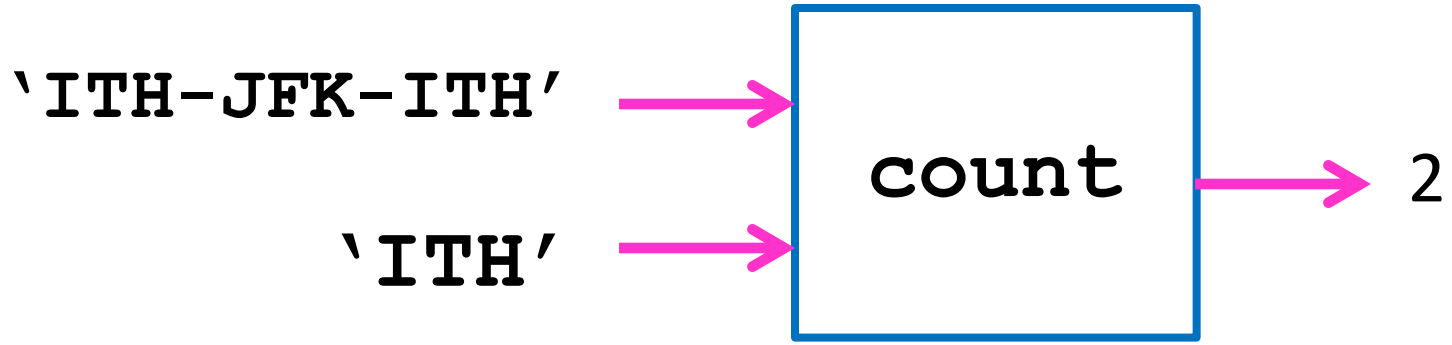
Three String Methods

`count`	How many times does string `t` occur in a string `s`?
`find`	Where is the first occurrence of string `t` in a string `s`?
`replace`	In a string `s` replace all occurrences of a string `s1` with a string `s2`.

There will be others.

Possible Designs

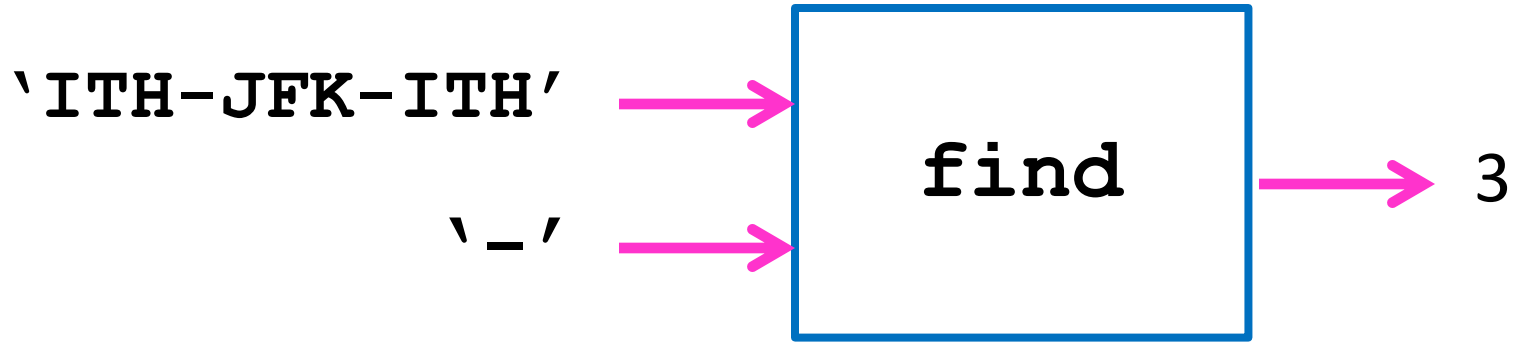
`count`	How many times does string `t` occur in a string `s`?



A function with two parameters. ??? `n = count(t,s)` ???

Possible Designs

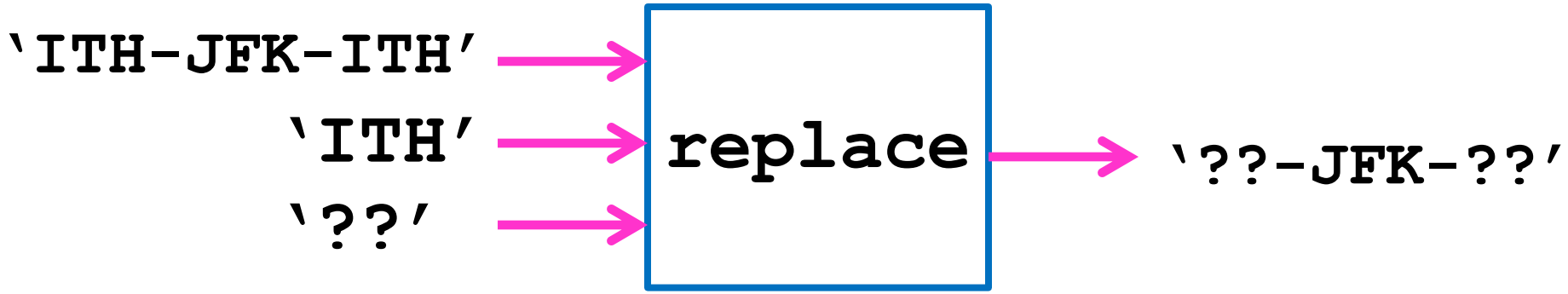
`find`	Where is the first occurrence of string `t` in a string `s`?



A function with two parameters. ??? `n = find(t,s)` ???

Possible Designs

replace In a string **s** replace all occurrences of a string s1 with a string **s2**.



A function with three parameters. ???? `sNew = replace(s,s1,s2)` ???

Methods: The Notation

Suppose

```
x = 'ITH-JFK-ITH'
y = 'ITH'
```

Instead of the usual function-call syntax

```
n = count(y,x)
```

we will write

```
n = x.count(y)
```

Methods: The Notation

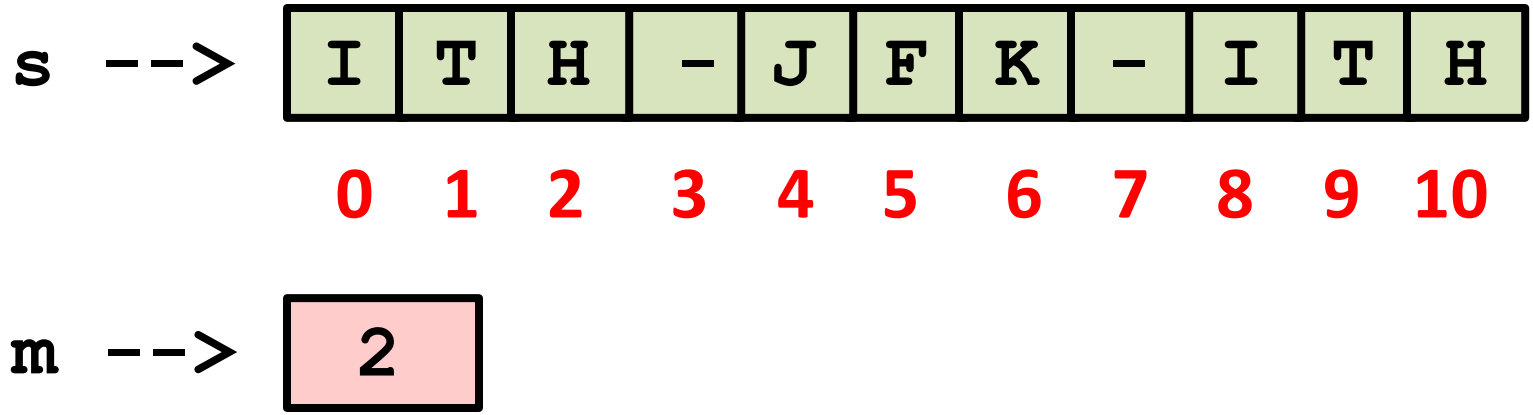
Here is the syntax associated with using a string method:

name of string . name of method (arg1,arg2,…)

Once again, the 'dot" notation

String Methods: count

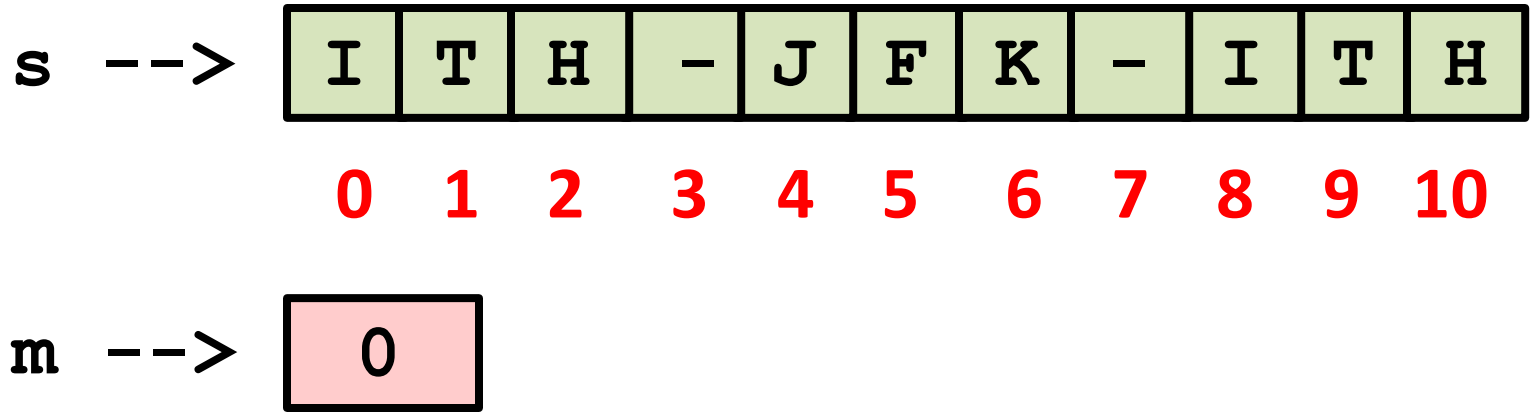
```
>>> s ='ITH-JFK-ITH'
>>> m = s.count('ITH')
```



`s1.count(s2)` the number of occurrences of string `s2` in string `s1`

String Methods: count

```
>>> s ='ITH-JFK-ITH'
>>> m = s.count('LGA')
```



`s1.count(s2)` the number of occurrences of string `s2` in string `s1`

count
The Formal Definition

If `s1` and `s2` are strings, then

`s1.count(s2)`

returns an int value that is the number of occurrences of string `s2` in string `s1`.

Note, in general `s1.count(s2)` is not the same as `s2.count(s1)`

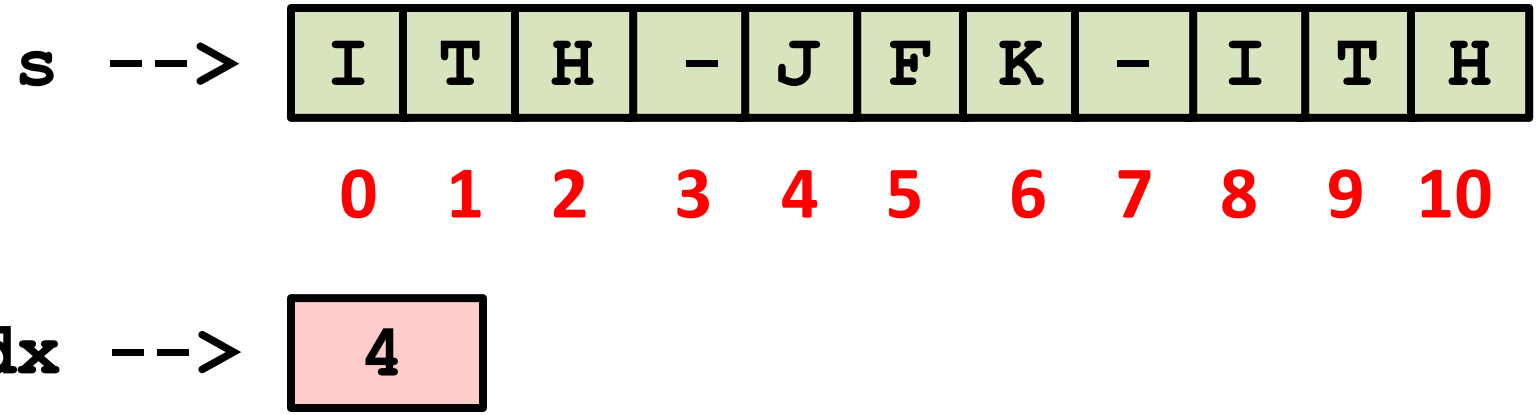
Using count: An Example

```
# Count the number of vowels…
A = 'auric goldfinger'
n = 0
n = n + A.count('a')
n = n + A.count('e')
n = n + A.count('i')
n = n + A.count('o')
n = n + A.count('u')
print n
```

Illegal: n = A.count('a' or 'e' or 'I' or 'o' or 'u')

String Methods: find

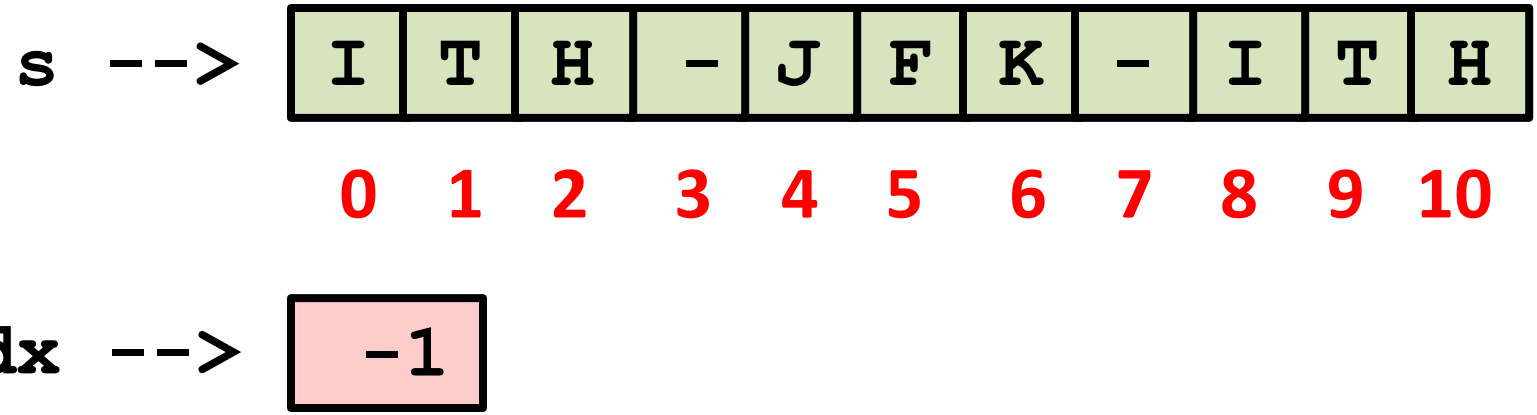
```
>>> s ='ITH-JFK-ITH'
>>> idx = s.find('JFK')
```



`s1.index(s2)` the index of the first occurrence of string `s2` in string `s1`

String Methods: find

```
>>> s ='ITH-JFK-ITH'
>>> idx = s.find('RFK')
```



`s1.index(s2)` evaluates to -1 if there is no occurrence of `s2` in `s1`

`find`
The Formal Definition

If `s1` and `s2` are strings, then

```
s1.find(s2)
```

returns an int value that is the index of the first occurrence of string `s2` in string `s1`.

If there is no such occurrence, then the value -1 is returned.

Using `find` : Some Examples

```
s = 'nine one one'
n1 = s.find('one')
n2 = s.find('two')
n3 = s.find(' nine')
```

n1 -> 5 n2 -> -1 n3 -> -1

The replace Method

```
s = 'one hundred and one'
t = s.replace(' ','-')
```

```
s ->  'one hundred and one'
```

```
t ->  'one-hundred-and-one'
```

Replacing one character with another

The replace Method

```
s = 'one hundred and one'
t = s.replace(' ','')
```

```
s ->  'one hundred and one'
```

```
t ->  'onehundredandone'
```

The null string has length 0.

Replacing each blank with the "null string"

The replace Method

```
s = 'one hundred and one'
t = s.replace('x','-')
```

```
s ->  'one hundred and one'
```

```
t ->  'one hundred and one'
```

No change if the character to be replaced is missing

The replace Method

```
s = 'one hundred and one'
t = s.replace('one','seven')
```

```
s ->  'one hundred and one'
```

```
t ->  'seven hundred and seven'
```

Replacing one substring with another

The replace Method

```
s = 'one hundred and one'
t = s.replace('two','seven')
```

```
s ->  'one hundred and one'
```

```
t ->  'one hundred and one'
```

No change if the designated substring is missing

`replace`
The Formal Definition

If `s`, `s1` and `s2` are strings, then

`s.replace(s1,s2)`

returns a copy of the string `s` in which every non- overlapping occurrence of the string `s1` is replaced by the string `s2`.

If `s1` is not a substring of `s`, then the returned string is just a copy of `s`.

Using replace : Some Examples

```
s = 'xxx'
t1 = s.replace('x','o')
t2 = s.replace('xx','o')
t3 = s.replace('xx','oo')
```

t1 -> 'ooo'

t2 -> 'ox'

t3 -> 'oox'

Replace does Not Replace

`s.replace(s1,s2)` does not change the value of `s`.

It produces a copy of `s` with the specified replacements.

You are allowed to overwrite the "original" `s` with the its "updated" copy:

```
s = s.replace(s1,s2)
```

Illegal!

```
s = 'abcdefgh'
s[5] = 'x'
```

Strings are immutable. They cannot be changed.

Have to ``live with'' the replace function, slicing, and concatenation

```
s = 'abcdefgh'
S = s[:5]+'x'+s[6:]
```

Upper and Lower Methods

```
s = 'A2sh?'
t1 = s.upper()
t2 = s.lower()
```

```
s ->  'A2sh?'
t1 ->  'A2SH?'
t2 ->  'a2sh?'
```

Boolean-Valued Methods

These methods return either **`True`** or **`False`**:

`islower()`

`isupper()`

`isalnum()`

`isalpha()`

`isdigit()`

Boolean-Valued Methods

	s='ab3?'	s='AbcD'	s='AB3'
s.islower()	True	False	False
s.isupper()	False	False	True

Boolean-Valued Methods

	'23'	'5a7'	'ab'	'-2.3'
s.isalnum()	True	True	True	False
s.isalpha()	False	False	True	False
s.isdigit()	True	False	False	False

Useful String Constants

```
alpha = string.letters
```

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Useful String Constants

```
specialChar = string.punctuation
```

```
!"#$%&'()*+,./:;<=>?@[\]^_`{|}~
```

Useful String Constants

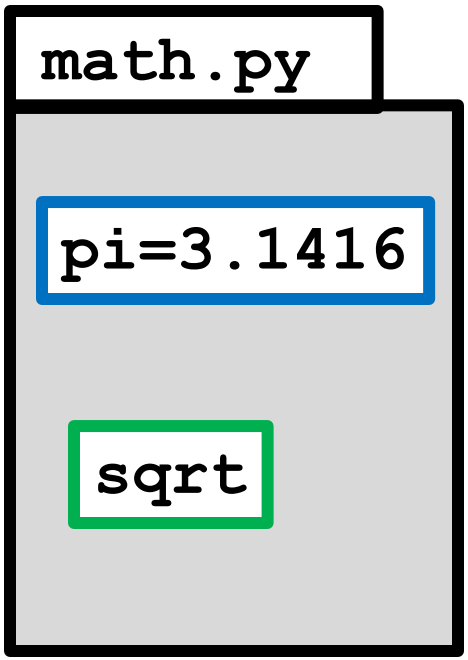
```
TheDigits  = string.digits
```

```
1234567890
```

The "Dot" Notation--Again

We have seen it with modules and import

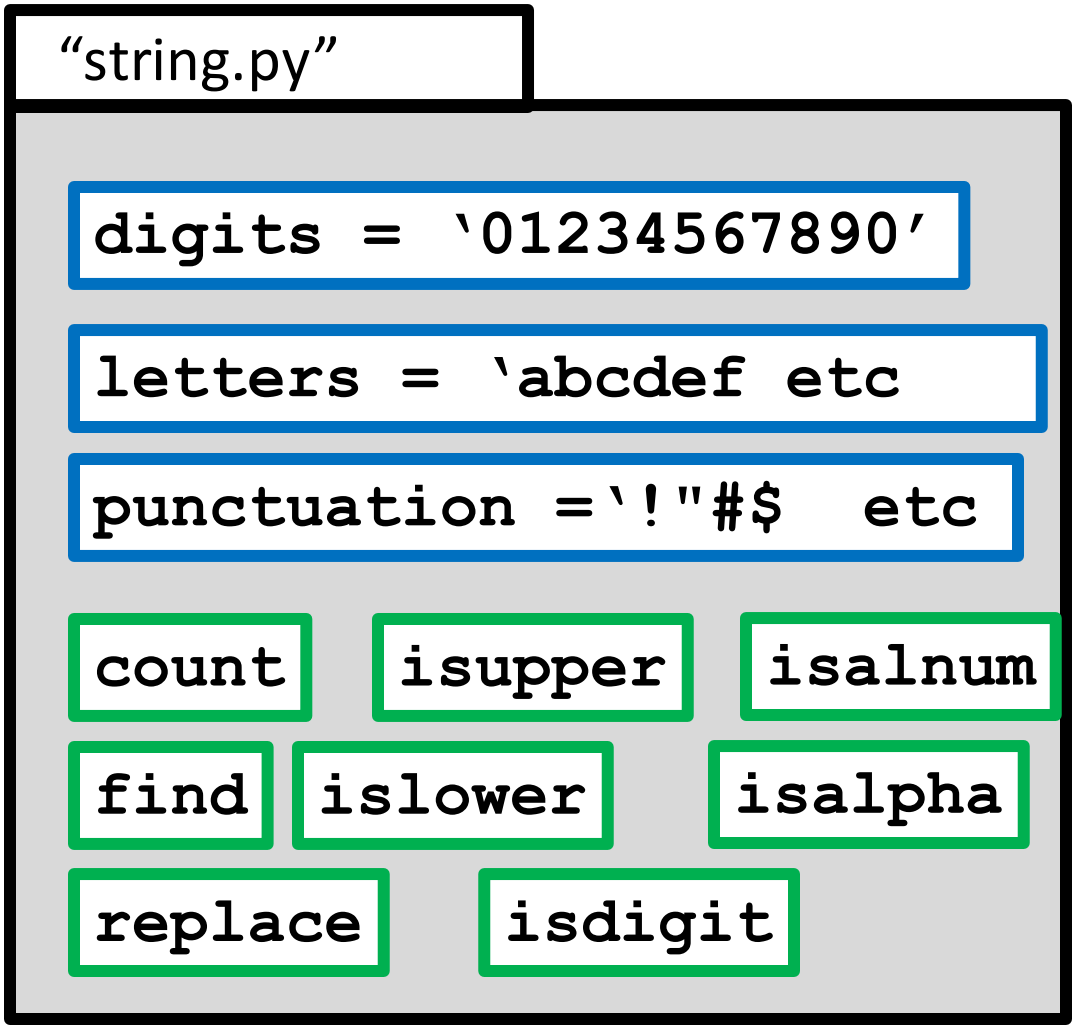
```
math.sqrt
math.pi
```



The "folder metaphor.

The "dot" means "go inside and get this"

String is a "Special" Module



The "folder" metaphor.

The "dot" means "go inside and get this"

`string` is actually a "class". More in a few lectures.

Iterating Through a String

Two problems we cannot easily solve:

1. Given a string `s`, assign to `t` the "reversed" string. `'abcd'` → `'dcba'`

2. Given a string `s`, how many digit characters does it contain? `'1or2or3'` → `3`

The Reverse String Problem

```
s =  'abcd'
t = ''
for c in s:
    t = c + t
```

s -> 'abcd'

t -> 'dcba'

How does the `for` loop work?

The Number-of-Digits Problem

```
s =   '2x78y'
n = 0
for c in s:
    if c.isdigit():
        n=n+1
```

s -> '2x78y'

n -> 3

How does the **for** loop work?

Using for to Traverse a String Character-by-Character

```
s = 'abcd'
for c in s:
    print c
```

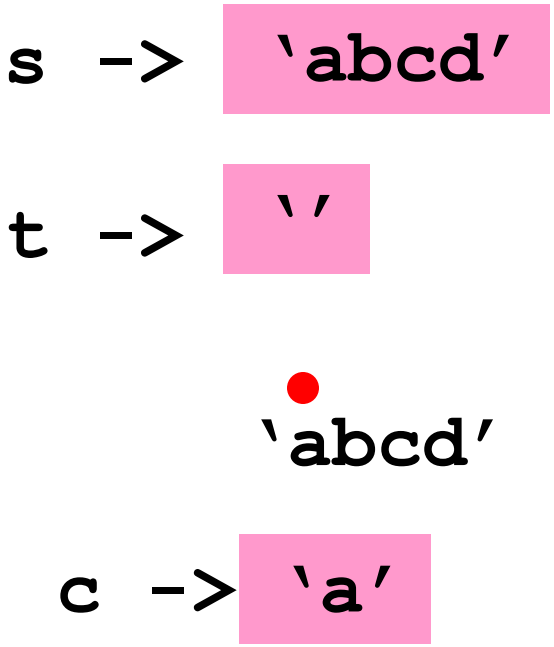
Output:

```
a
b
c
d
```

In this example, the "for-loop" variable is `c`. One at a time, it takes on the value of each character in s.

The Reverse String Problem

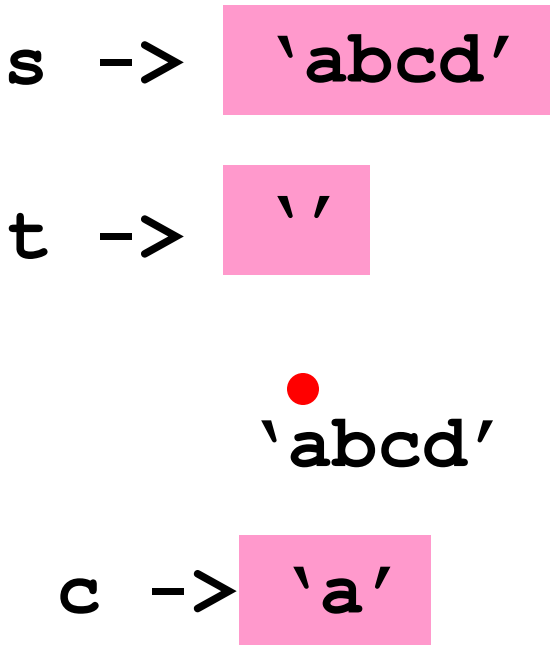
```
s =  'abcd'
t = ''
for c in s:
    t = c + t
print t
```



At the start of the loop, c is assigned the zeroth character in s.

The Reverse String Problem

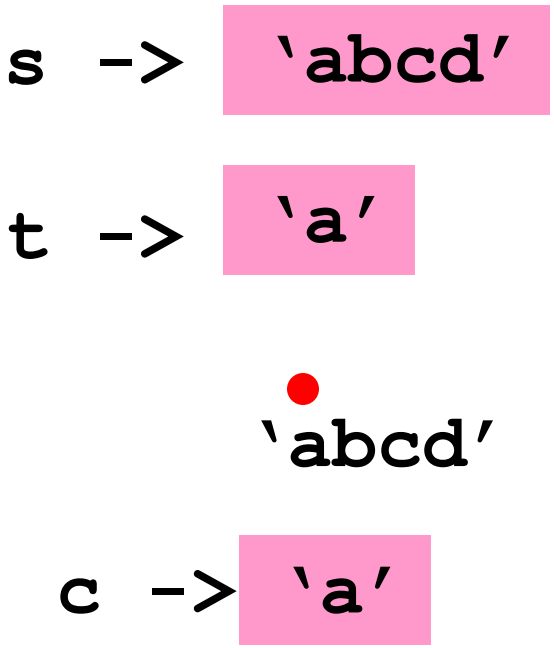
```
s = 'abcd'
t = ''
for c in s:
    t = c + t
print t
```



The loop body is executed using that value in c.

The Reverse String Problem

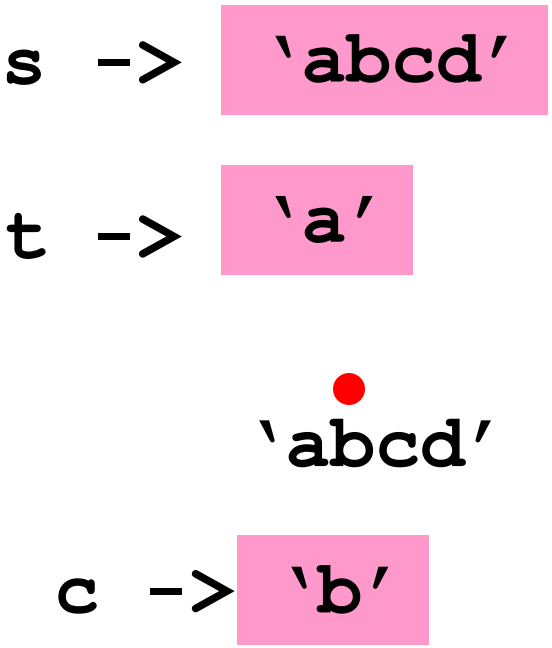
```
s =  'abcd'
t = ''
for c in s:
    t = c + t
print t
```



The loop body is executed using that value in c.

The Reverse String Problem

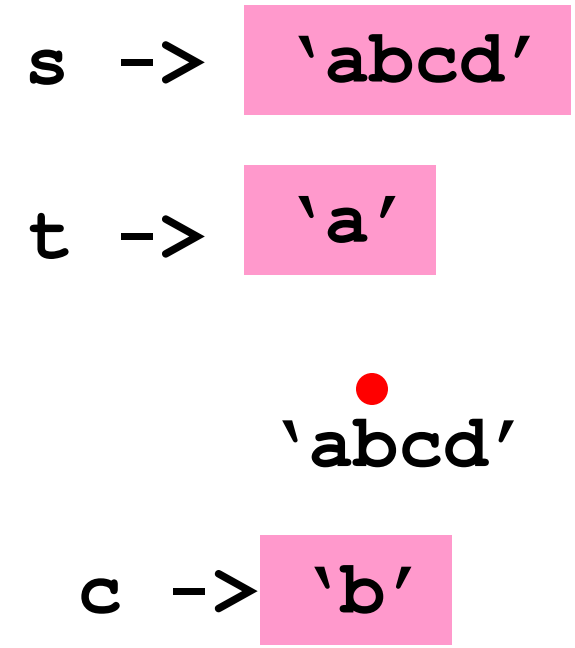
```
s =  'abcd'
t = ''
for c in s:
    t = c + t
print t
```



The next time through the loop, c is assigned the first character in s.

The Reverse String Problem

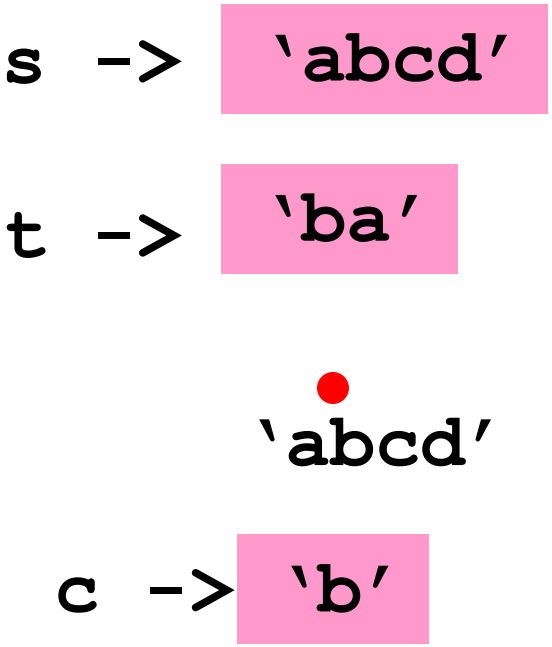
```
s = 'abcd'
t = ''
for c in s:
    t = c + t
print t
```



The loop body is executed using that value in c.

The Reverse String Problem

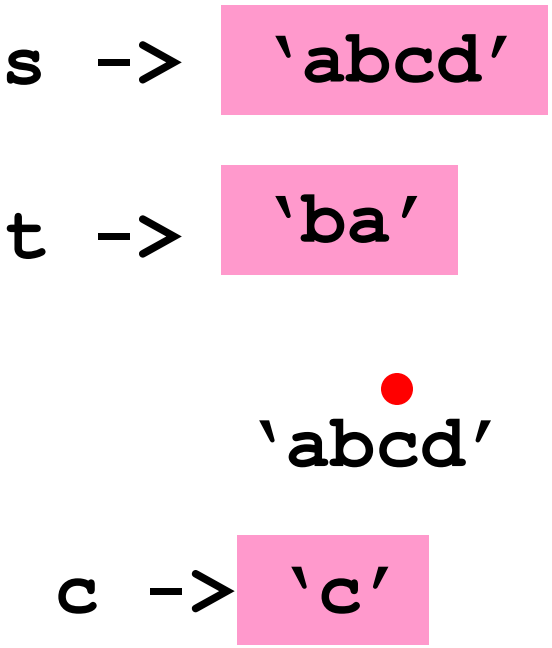
```
s = 'abcd'
t = ''
for c in s:
    t = c + t
print t
```



The loop body is executed using that value in c.

The Reverse String Problem

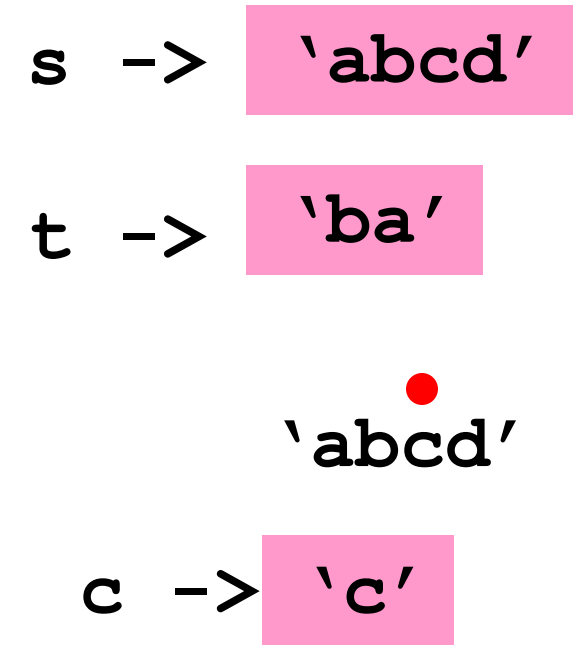
```
s =  'abcd'
t = ''
for c in s:
    t = c + t
print t
```



The next time through the loop, c is assigned the second character in s.

The Reverse String Problem

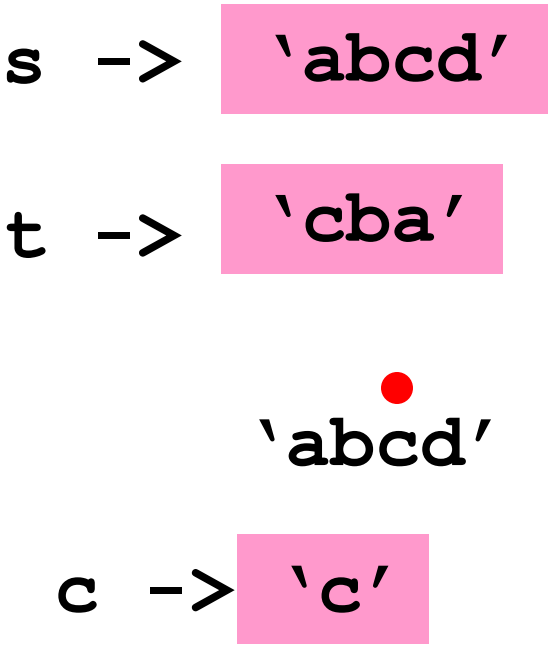
```
s =  'abcd'
t = ''
for c in s:
    t = c + t
print t
```



The loop body is executed using that value in c.

The Reverse String Problem

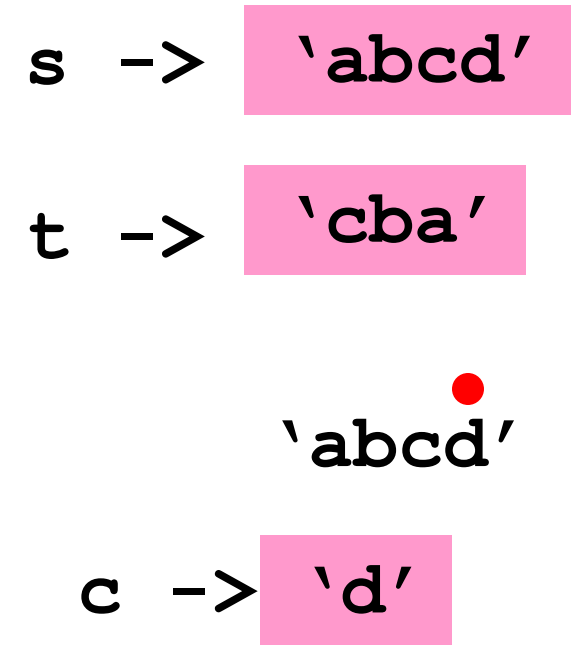
```
s = 'abcd'
t = ''
for c in s:
    t = c + t
print t
```



The loop body is executed using that value in c.

The Reverse String Problem

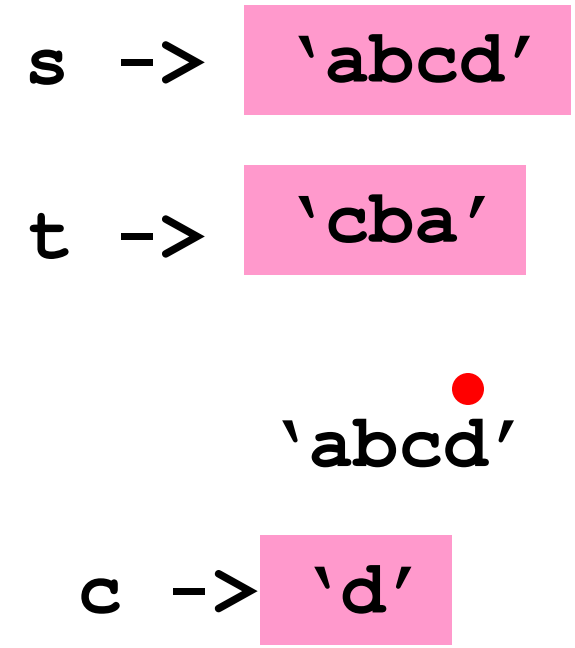
```
s =  'abcd'
t = ''
for c in s:
    t = c + t
print t
```



The last time through the loop, c is assigned the third character in s.

The Reverse String Problem

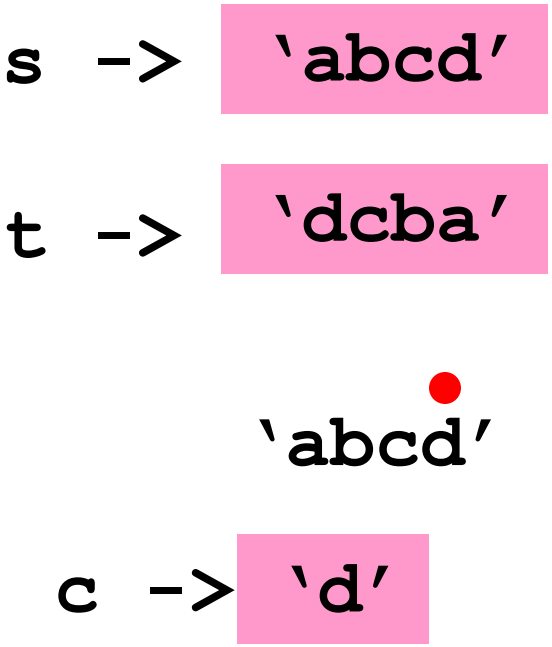
```
s = 'abcd'
t = ''
for c in s:
    t = c + t
print t
```



The loop body is executed using that value in c.

The Reverse String Problem

```
s = 'abcd'
t = ''
for c in s:
    t = c + t
print t
```



The loop body is executed using that value in c.

The Reverse String Problem

```
s =  'abcd'
t = ''
for c in s:
    t = c + t
print t
```

s -> 'abcd'

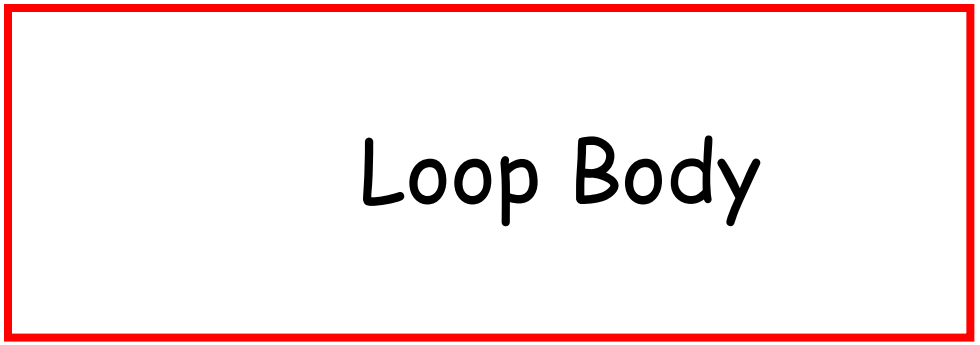
t -> 'dcba'

Output: dcba

The string has been traversed. The iteration ends. The next statement after the loop is executed. Indentation important.

for-loop Mechanics

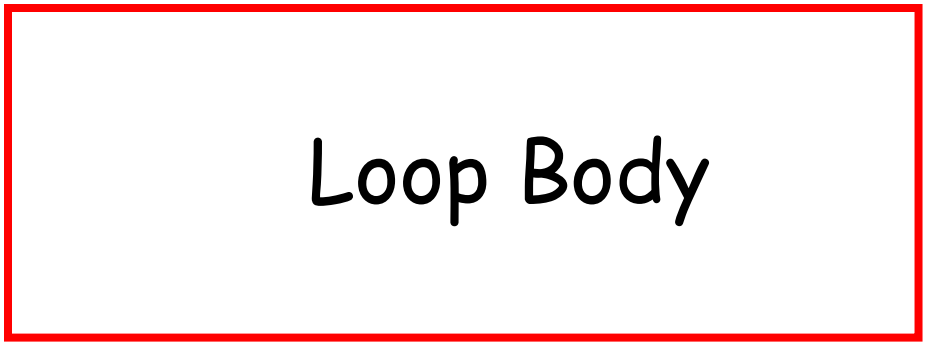
```
for <loop variable> in <string>:
```



If the string has length `n`, then the loop body is executed `n` times.

for-loop Mechanics

```
for x in y:
```



Let `x = y[0]` and then execute the loop body.
Let `x = y[1]` and then execute the loop body.
Let `x = y[2]` and then execute the loop body.
etc
Let `x = y[n-1]` and then execute the loop body.

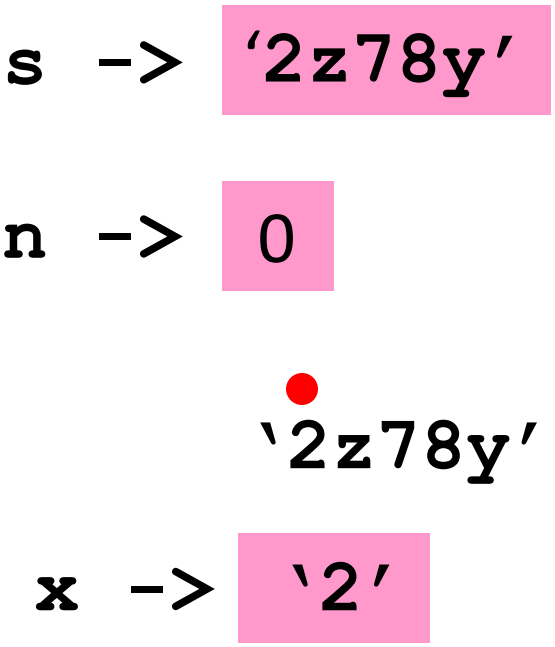
The Number-of-Digits Problem

Given a string s, how many of its characters are digit characters?

`'a10b20c30d40'` → `8`

The Number-of-Digits Problem

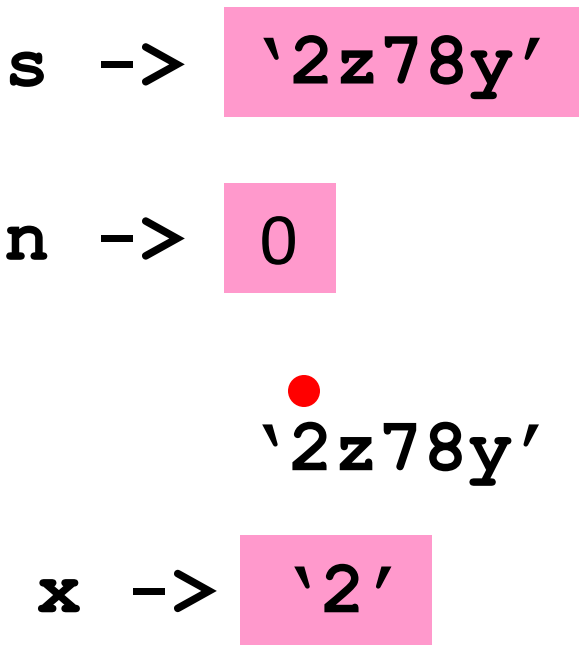
```
s =  '2z78y'
n = 0
for x in s:
    if x.isdigit():
        n=n+1
print n
```



At the start of the loop, **x** is assigned the zeroth character in **s**.

The Number-of-Digits Problem

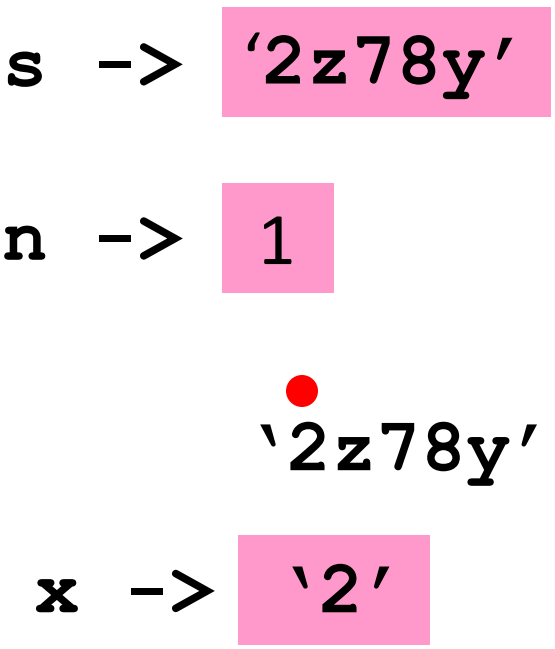
```
s =  '2z78y'
n = 0
for x in s:
    if x.isdigit():
        n=n+1
print n
```



The loop body is executed using that value in **x**.

The Number-of-Digits Problem

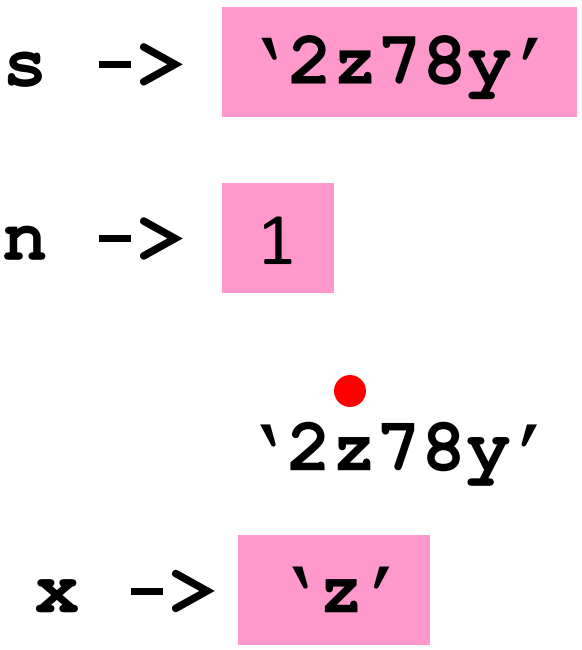
```
s =  '2z78y'
n = 0
for x in s:
    if x.isdigit():
        n=n+1
print n
```



The loop body is executed using that value in **x**.

The Number-of-Digits Problem

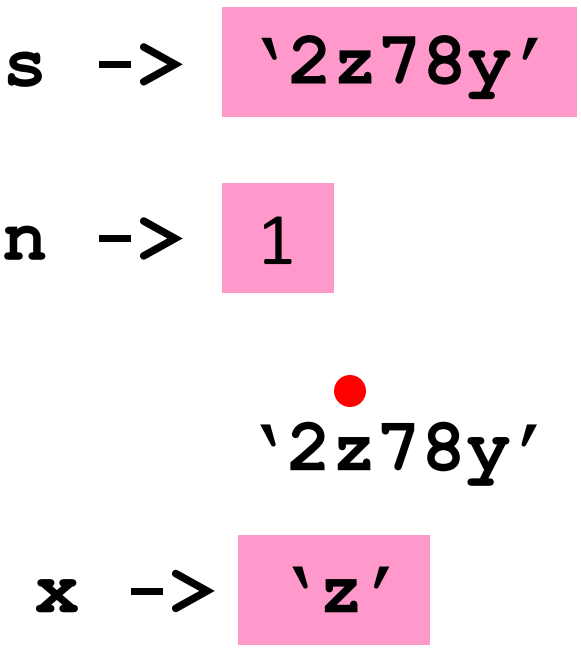
```
s =  '2z78y'
n = 0
for x in s:
    if x.isdigit():
        n=n+1
print n
```



The next time through the loop, **x** is assigned the first character in **s**.

The Number-of-Digits Problem

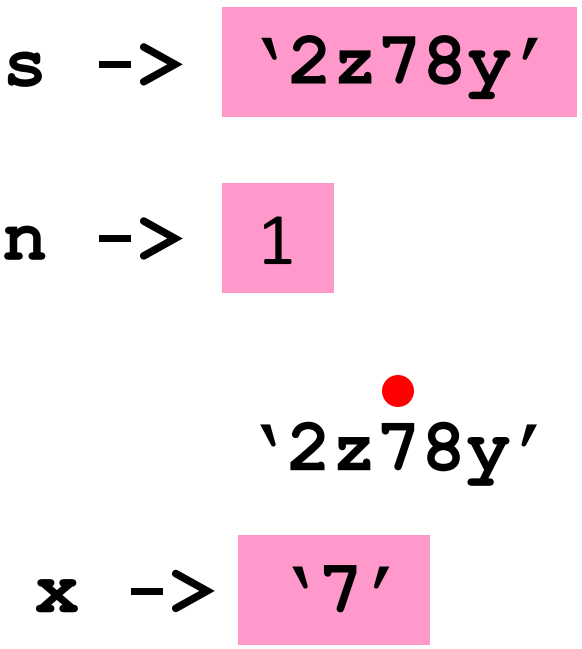
```
s =  '2z78y'
n = 0
for x in s:
    if x.isdigit():
        n=n+1
print n
```



The loop body is executed using that value in **x**.

The Number-of-Digits Problem

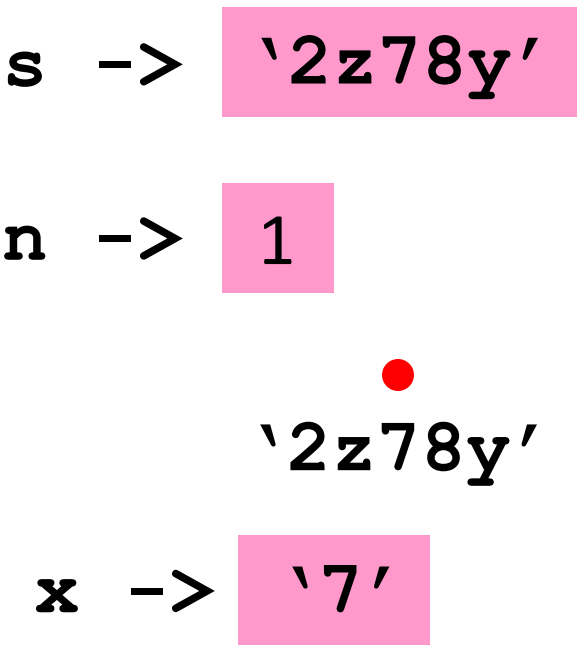
```
s =  '2z78y'
n = 0
for x in s:
    if x.isdigit():
        n=n+1
print n
```



The next time through the loop, **x** is assigned the second character in **s**.

The Number-of-Digits Problem

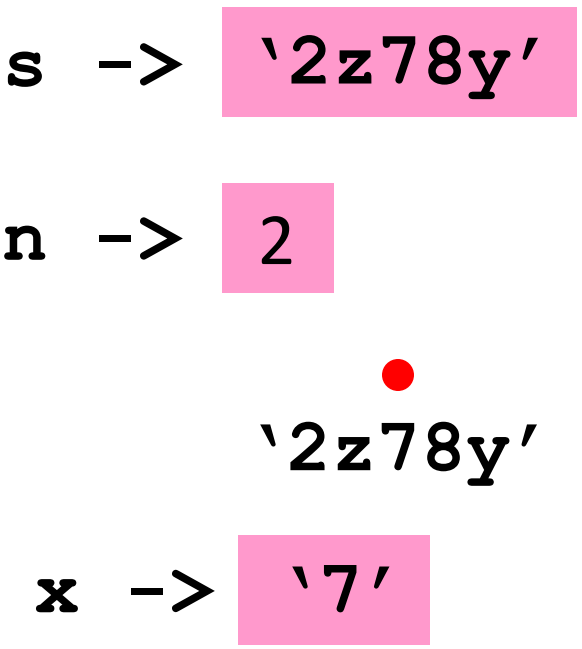
```
s =  '2z78y'
n = 0
for x in s:
    if x.isdigit():
        n=n+1
print n
```



The loop body is executed using that value in **x**.

The Number-of-Digits Problem

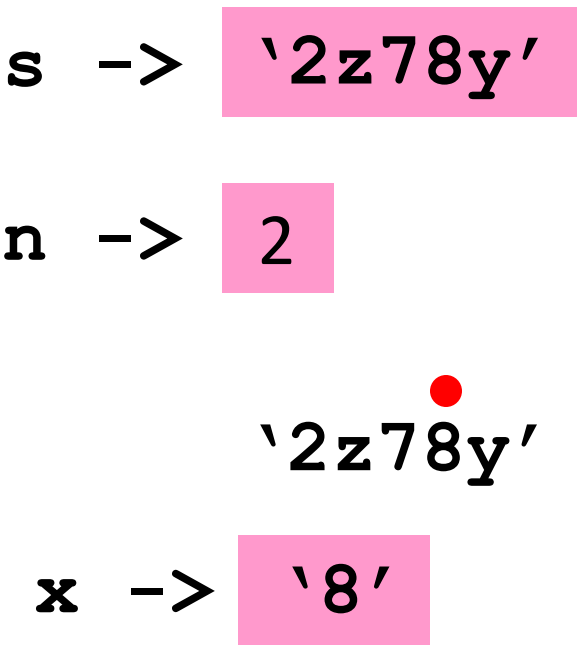
```
s =  '2z78y'
n = 0
for x in s:
    if x.isdigit():
        n=n+1
print n
```



The loop body is executed using that value in **x**.

The Number-of-Digits Problem

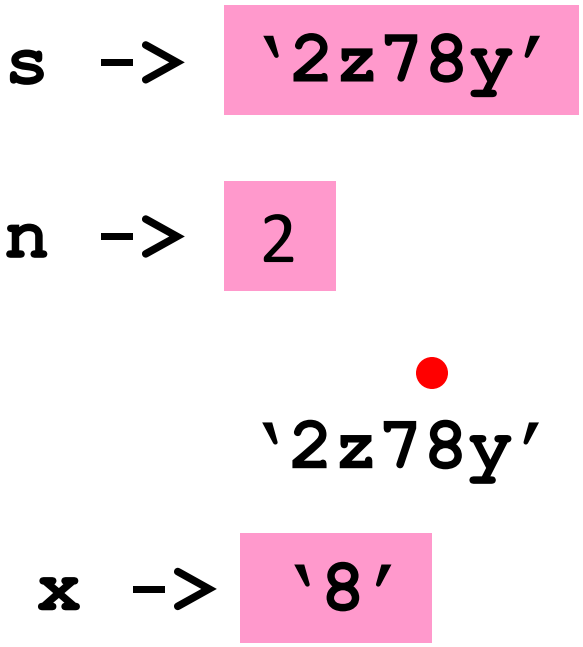
```
s =  '2z78y'
n = 0
for x in s:
    if x.isdigit():
        n=n+1
print n
```



The next time through the loop, **x** is assigned the third character in **s**.

The Number-of-Digits Problem

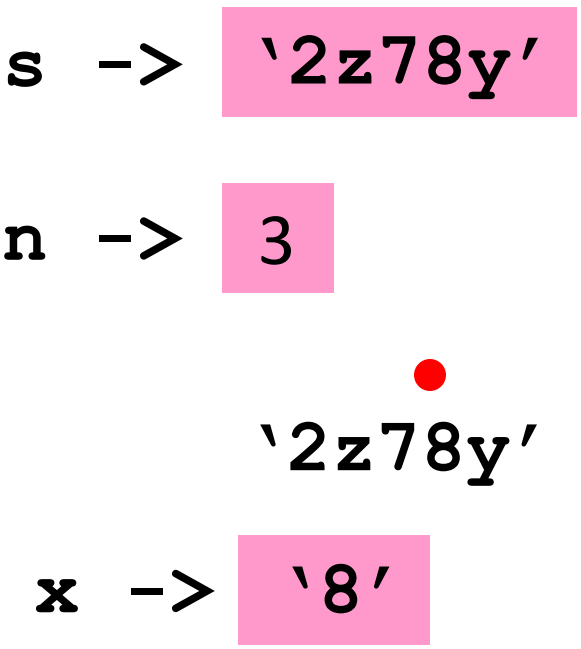
```
s =  '2z78y'
n = 0
for x in s:
    if x.isdigit():
        n=n+1
print n
```



The loop body is executed using that value in **x**.

The Number-of-Digits Problem

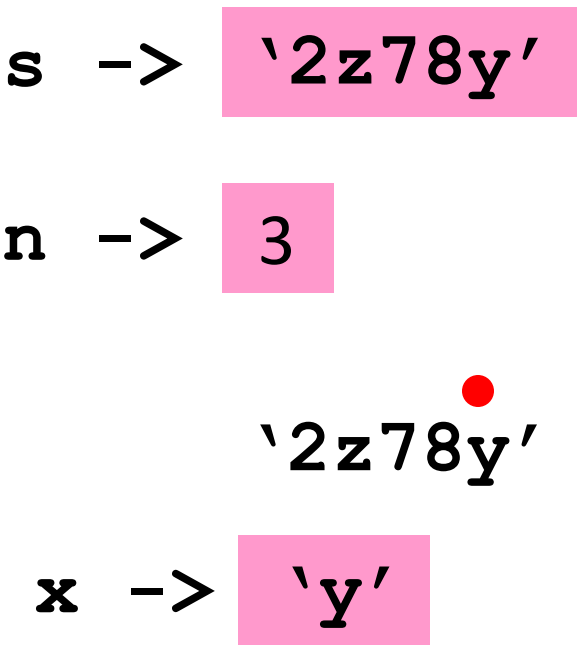
```
s =  '2z78y'
n = 0
for x in s:
    if x.isdigit():
        n=n+1
print n
```



The loop body is executed using that value in **x**.

The Number-of-Digits Problem

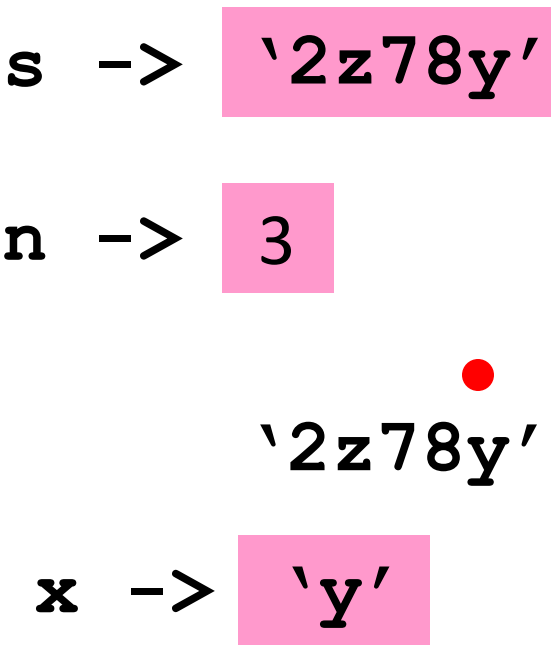
```
s =  '2z78y'
n = 0
for x in s:
    if x.isdigit():
        n=n+1
print n
```



The next time through the loop, **x** is assigned the fourth character in **s**.

The Number-of-Digits Problem

```
s =  '2z78y'
n = 0
for x in s:
    if x.isdigit():
        n=n+1
print n
```



The loop body is executed using that value in **x**.

The Number-of-Digits Problem

```
s =   '2z78y'
n = 0
for x in s:
    if x.isdigit():
        n=n+1

print n
```

s -> '2z78y'

n -> 3

Output:

3

The string has been traversed. The iteration ends. The next statement after the loop is executed. Indentation important.

Function for Reversing Strings

```
def Reverse(s):
    """ Returns a string that is obtained
    from s by reversing the order of its
    characters.

    Precondition: s is a string."""

    t = ''          # The empty string
    for c in s:
        t = c+t     # Repeated concatenation
    return t
```

Function for Counting Digits

```
def nDigits(s):
    """ Returns an int whose value is the
    number of digit characters that are in
    s.

    Precondition: s is a string."""
    n = 0;
    for c in s:
        # Increment n if c is a digit
        if c.isdigit():
            n=n+1
    return n
```