Objectives

- Finish Animation
- Strings

Review: OO Programming

- How do we create an object?
- How do we know what methods are available to us for an object of a certain type?
- What is a benefit of OO programming?

Problem: Animate Moving to User Click

- Use combinations of the method **`move`** and the function **`sleep`**
  - Need to **sleep** so that humans can see the graphics moving
  - Computer would process the **move**s too fast!
- **`sleep`** is part of the **`time`** module
  - Takes a `float` parameter representing *seconds* and pauses for that amount of time
- **Problem**: In X steps, move from the circle's current location to the location clicked by user

`circleShiftAnim.py`

Text Processing

- Mostly focused on numbers so far
  - A little on graphics
- We can manipulate strings to do useful work
  - Web search: finding most relevant documents to a query
  - Analyzing web logs (who is looking at my web page?)
  - Many, many others
- **Today's Focus**: the **`str`** data type and what you can do with them

String Comparisons

- Same operations as with numbers:
  - ==, !=
  - <, <=
  - >, >=

Alphabetical comparison

- Use in conditions for `if` statements and `while` loops

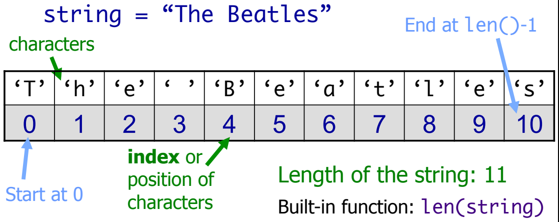
```
if userpick == pick4num:
    print "We have a winner!"
else:
    print "You lose."
```

`string_compare.py`

Strings

- Actually a *sequence* of characters
  - Example:

`string = “The Beatles”`



‘T’	‘h’	‘e’	‘ ’	‘B’	‘e’	‘a’	‘t’	‘l’	‘e’	‘s’
0	1	2	3	4	5	6	7	8	9	10

Length of the string: 11

Built-in function: `len(string)` to find length of a string

Iterating Through a String

- Use a **for** loop to iterate through *characters* in a string

string of length 1

```
for char in string:
    print char
```

  - Read as "for each character in the string"

Python shell

Substrings Operator: []

Literally, **not** optional

- Look at a particular character in the string
  - Syntax: `str[<integer_expression>]`
  - [Positive value]: index of character
  - [Negative value]: count backwards from end
- Examples:
  - <sequence>[0] returns the first element/char
  - <sequence>[-1] returns the last element/char

We will deal with sequences beyond strings later.

Examples in shell

Substrings Operator: []

- Look at a particular character in the string
  - Syntax: `string[<integer_expression>]`
- Examples with `string = "The Beatles"`

T	h	e		B	e	a	t	l	e	s
0	1	2	3	4	5	6	7	8	9	10

Expression	Result
`string[0]`	
`string[3]`	
`string[len(str)]`	
`string[len(str)-1]`	
`string[-1]`	

Substrings Operator: []

- Look at a particular character in the string
  - Syntax: `string[<integer expression>]`
- Examples with `string = "The Beatles"`

T	h	e		B	e	a	t	l	e	s
0	1	2	3	4	5	6	7	8	9	10

Expression	Result
`string[0]`	"T"
`string[3]`	" "
`string[len(str)]`	IndexError
`string[len(str)-1]`	"s"
`string[-1]`	"s"

Iterating Through a String

- Alternatively, can iterate through the *positions* in a string
  - Could write as a **while** loop as well

An integer

```
for pos in xrange(len(string)):
    print string[pos]
```

Index into the string

`whilestr.py`

Summary: Iterating Through a String

- For each character in the string

string of length 1

```
for char in string:
    print char
```

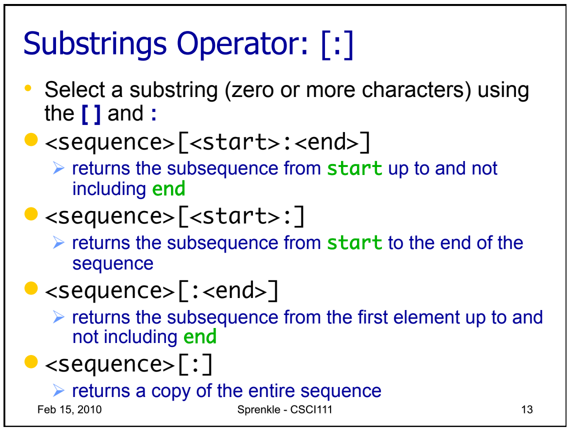
Determines loop's behavior

- For each position in the string

An integer

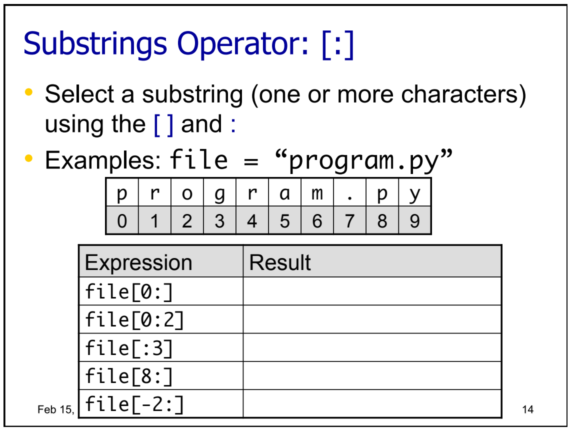
```
for pos in xrange(len(string)):
    print string[pos]
```

Index into the string



Substrings Operator: [:]

- Select a substring (zero or more characters) using the **[ ]** and **:**
- `<sequence>[<start>:<end>]`
  - returns the subsequence from `start` up to and not including `end`
- `<sequence>[<start>:]`
  - returns the subsequence from `start` to the end of the sequence
- `<sequence>[:<end>]`
  - returns the subsequence from the first element up to and not including `end`
- `<sequence>[:]`
  - returns a copy of the entire sequence

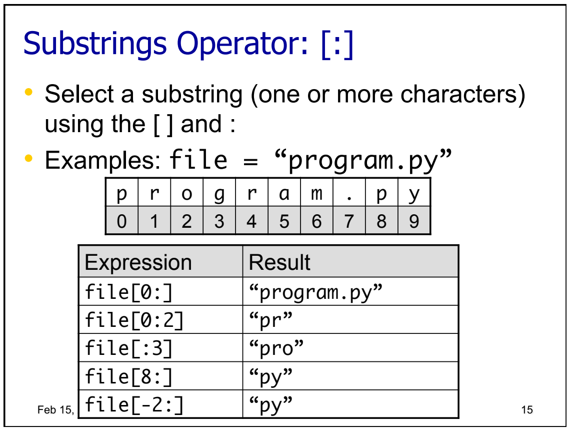


Substrings Operator: [:]

- Select a substring (one or more characters) using the [ ] and :
- Examples: `file = “program.py”`

p	r	o	g	r	a	m	.	p	y
0	1	2	3	4	5	6	7	8	9

Expression	Result
`file[0:]`	
`file[0:2]`	
`file[:3]`	
`file[8:]`	
`file[-2:]`	

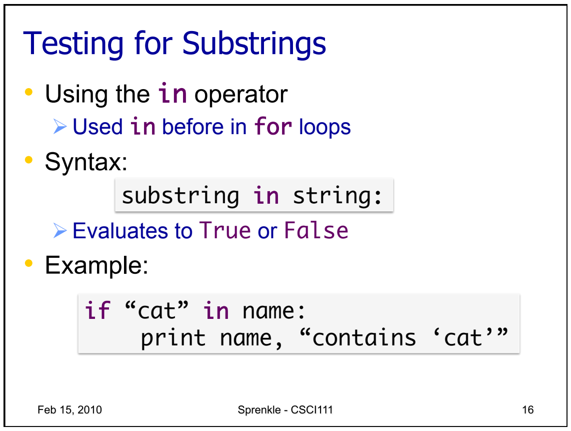


Substrings Operator: [:]

- Select a substring (one or more characters) using the [ ] and :
- Examples: `file = “program.py”`

p	r	o	g	r	a	m	.	p	y
0	1	2	3	4	5	6	7	8	9

Expression	Result
`file[0:]`	`“program.py”`
`file[0:2]`	`“pr”`
`file[:3]`	`“pro”`
`file[8:]`	`“py”`
`file[-2:]`	`“py”`



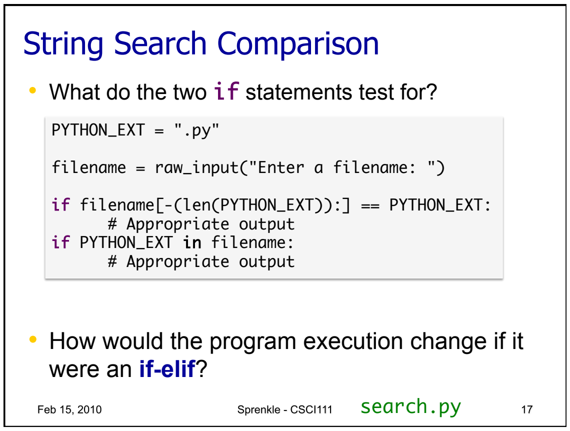
Testing for Substrings

- Using the `in` operator
  - Used `in` before in `for` loops
- Syntax:

```
substring in string:
```

  - Evaluates to `True` or `False`
- Example:

```
if “cat” in name:
    print name, “contains ‘cat’”
```



String Search Comparison

- What do the two `if` statements test for?

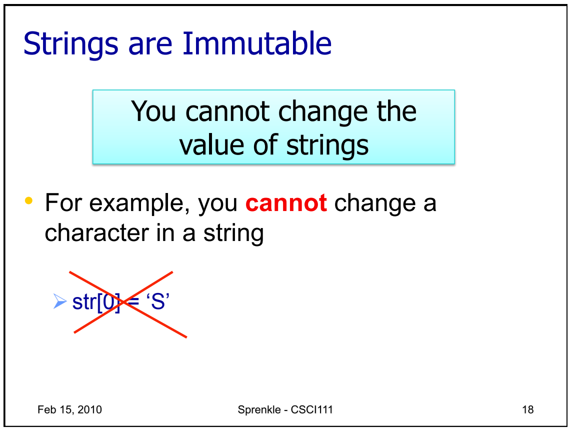
```
PYTHON_EXT = ".py"

filename = raw_input("Enter a filename: ")

if filename[-(len(PYTHON_EXT)):] == PYTHON_EXT:
      # Appropriate output
if PYTHON_EXT in filename:
      # Appropriate output
```

- How would the program execution change if it were an **if-elif**?

`search.py`



Strings are Immutable

You cannot change the value of strings

- For example, you **cannot** change a character in a string
  - ~~str[0] = ‘S’~~

Practice: Revised Pick4 Numbers

- Tell the user how many numbers they got right
  - Get prizes for having some numbers right
- Examples:

Pick4 Num	User's Pick	Num Correct
"7737"	"1234"	1
"0204"	"1234"	2
"1234"	"1234"	4

pick4num_places.py

This Week

- Tuesday lab
  - Due Friday
- Broader Issue: Excel Bug