

---

# CS 111: Program Design I

## Lecture 5: Strings & heading to Caesar

---

Robert H. Sloan & Richard Warner  
University of Illinois at Chicago  
September 10, 2019



# **ASSIGNMENT (CONT.)**

# Assignment to variables: Semantics

$\langle \text{variable} \rangle = \langle \text{expression} \rangle$

1. Evaluate  $\langle \text{expression} \rangle$
2. Put that value into computer's memory and attach name  $\langle \text{variable} \rangle$  as "sticky note" giving name for that memory location

---

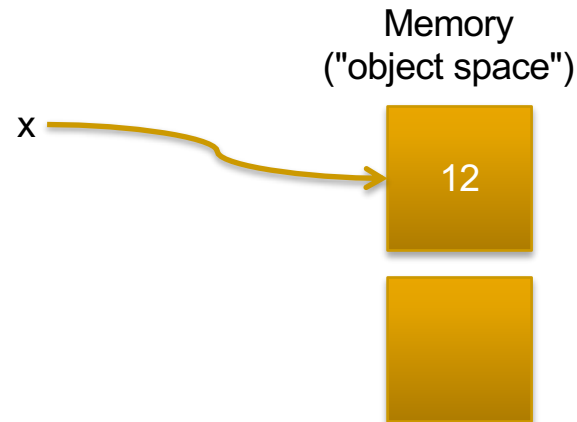

# Expressions

- Can be simple value, e.g.,
  - "Sandra Day O'Conner" or 17
- Also can be almost any mathematical statement

# Expressions

- Can be a simple value, like
  - "Sandra Day O'Conner" or 17
- Also can be almost any mathematical statement

$x = 6 * 2$   
 $y = x - 10$

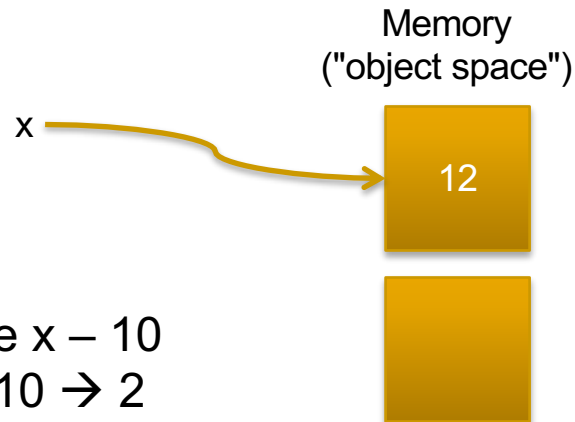


# Expressions

- Can be a simple value, like
  - "Sandra Day O'Conner" or 17
- Also can be almost any mathematical statement

$x = 6 * 2$

$y = x - 10$

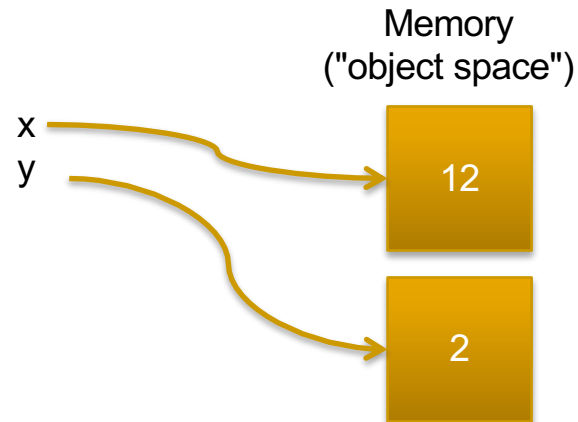


# Expressions

- Can be a simple value, like
  - "Sandra Day O'Conner" or 17
- Also can be almost any mathematical statement

$x = 6 * 2$

$y = x - 10$



---

At end of this code  $y$  will be

$$x = 5$$

$$y = x * 3$$

$$x = 2$$

A. 2

B. 5

C. 6

D. 15



---

At end of this code  $y$  will have what value?

$y = 3 + 2 + 1$

$y = y * 2$

- A. 2
- B. 6
- C. 8
- D. This code will cause an error
- E. 12

## Code

```
x = 7
print(x)
x = x + 1
print(x)
y = x - 3
print(x)
print(y)
```

At the end of running this code, what will appear from the print statements in the execution window?

```
7
8
5
5
```

**A**

```
7
8
8
5
```

**B**

```
7
7
7
4
```

**C**

**D. This will cause an error**

**E. I don't know**



# **A BIT MORE PYTHON FUNCTIONS**

---

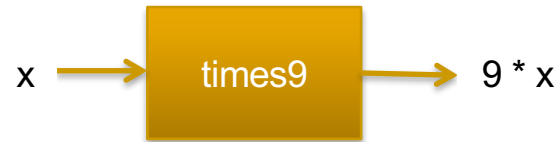
# Composition of functions

- Functions can use other functions!
- Both built-in Python functions and ones you write

# Composition example

```
def triple(x):  
    return 3 * x
```

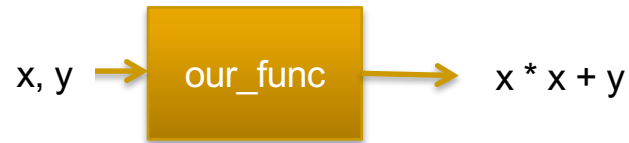
```
def times9(x):  
    return 9 * x
```



**OR**

```
def times9(x):  
    return triple(triple(x))
```

# Multiple inputs (or no inputs) ok



```
# our_func
# Authors: Bob and Richard
# Date: September 52, 2019

def our_func(x, y):
    """returns x squared plus y"""
    return x * x + y
```

---

# Order of common operations

- Things in ()s first. Use ()s whenever you are in the slightest doubt
- Next \*\* (exponentiation)
- Next \*, /, and //
- Lastly + and –

# What is printed

```
print(6 + 1 * 3)
```

- A. 9
- B. 21
- C. Some other value





**STRINGS, STRINGS, STRINGS!**

# Strings

- Are really key data type for many legal issues
  - **Legal analytics**, and **automated discovery** involve searches through zillions of pages of documents
    - And automation is putting some lawyers out of work!
  - **Privacy** in 2019 requires **encryption**
  - **Defense in depth** against bad consequences of data breaches in 2019 requires **encryption**
    - **And encryption works on strings**
  - **Web search** string based

# Python awesome language for working with strings

- Python happy to have strings in 'these' or "these" or even `"""these"""`, making it *much* simpler to deal with embedded quote characters such as
  - `'''Justice Sandra Day O'Conner wrote, "The power I exert on the court depends on the power of my arguments, not on my gender."'''`
- Python can handle carriage returns in strings

---

paragraph1 = "Many excellent books offer illuminating descriptions of the current crises in online privacy and security. We take the next step and offer solutions. Our solutions are public policy recommendations. Society needs innovative policies to reap the proper benefits from rapid technological change. We hope our recommendations will be adopted and be successful, but we also have another important goal: a shared understanding of the problems and a common language in which to discuss and analyze solutions. Finding adequate solutions to today's online privacy and security problems requires combining a computer scientist's expertise with a lawyer's understanding of how to forge sound public policy and laws. You don't need to be a legal scholar or to know any computer science to read this book, even though it contains sophisticated and accurate computer science and law. We have written as much as possible in plain English, but our plain English descriptions should be of interest even to experts. Solving the privacy and security problems means experts in one field have to find ways to communicate with experts in another."

# String

- **String**: any sequence of characters enclosed in single, double, or triple quotes
  - Beginning and ending quote marks need to match
  - Can use either 3 single quotes `'''` or 3 double quotes for triple quotes
  - Convention: For short-ish strings ( $\leq 40$  characters) use single quotes if no internal quotes or newlines
  - Convention: Docstrings are in triple quotes
- Example of a **sequence**
- Other important kind of sequence is list (coming soon)

---

# Which is a valid Python string?

- A. "Admiral Grace Hopper"
- B. "Admiral Grace Hopper'
- C. Admiral Grace Hopper
- D. "Admiral Grace Hopper' "
- E. Both A and D

# Things we can do with strings

- Find their length, using the built-in Python function `len`

```
>>> first_chief = 'Jay'
```

```
>>> len(first_chief)
```

```
3
```

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

```
8
```

- String arithmetic: `+`  $\rightarrow$  concatenation; `*`  $\rightarrow$  repeat

```
>>> "Chief Justice " + first_chief
```

```
'Chief Justice Jay'
```

```
>>> 4 * first_chief
```

```
'JayJayJayJay'
```

# Arithmetic limited to + and integer \*

```
>>> firstChief / 1
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

```
>>> 3.0 * firstChief
```

Also Barf

- Importance of types: floats are not integers, and multiplication of int by string makes some sense, but multiplication of float by string makes no sense



# What does this code print?

```
x = 'Go'  
y = 'Flames!'  
print(x + y)
```

- A. Go Flames!
- B. GoFlames!
- C. Error because the addition is illegal
- D. Go

# Indexing strings

- `[i]` after string gives character number `i`
- **Important:** Python (and many computer scientists!) counts from 0, not from 1!
- So if `first_justice = 'Jay'`
  - `first_justice[0]` → "J"
  - `first_justice[1]` → "a"
  - `first_justice[2]` → "y"
  - `first_justice[3]` → Out of range error!
  - `first_justice[7]` → also out of range error

# Negative indices

- Sometimes we want to get our hands on the *last* character of a string
- One way: `first_justice[len(first_justice) - 1]`
  - *Why is that - 1 in there?*

# Negative indices

- Sometimes we want to get our hands on the *last* character of a string
- One way: `first_justice[len(first_justice) - 1]`
  - *Why is that - 1 in there?*
  - Because "Jay" has 3 characters, so length 3, but characters numbered 0, 1, 2, and in general the characters of string `s` are numbered
  - 0, 1, ..., `len(s) - 1`

# Negative indices

- Sometimes we want to get our hands on the *last* character of a string
- One way: `first_justice[len(first_justice) - 1]`
- Another, often easier to think about way:
  - Index -1 is always *last character* of string
  - And index -2 2<sup>nd</sup> last character, and so on
  - So `first_justice[-1]` → 'y'; `first_justice[-3]` → 'J'

# Summary: Indexing (including negative)

0 1 2 3 4 5 6 7 8 9 1 1 1 1 1 1 1  
0 1 2 3 4 5 6

```
>>> s = "Register to vote!"
```

```
>>> len(s)
```

```
17
```

```
>>> s[7]
```

```
'r'
```

```
>>> s[17]
```

```
error!
```

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

```
'!'
```

```
>>> s[-1]
```

```
'!'
```

# Finding character's (or substring's) index

- Use String method **find** to obtain index of first occurrence of item in string or -1 if not in string at all

```
In [1]: hi = 'hello world'
```

```
In [2]: hi.find('h')
```

```
Out[2]: 0
```

```
In [3]: hi.find('o')
```

```
Out[3]: 4
```

```
In [4]: hi.find('q')
```

```
Out[4]: -1
```

# This will print?

```
hi = 'Hello World'  
hey = 'Hey world!'  
print(hi.find('o'), hey.find('o'))
```

- A. 4 4
- B. 4 5
- C. 5 5
- D. 5 6



---

`find()` String method works for substrings too

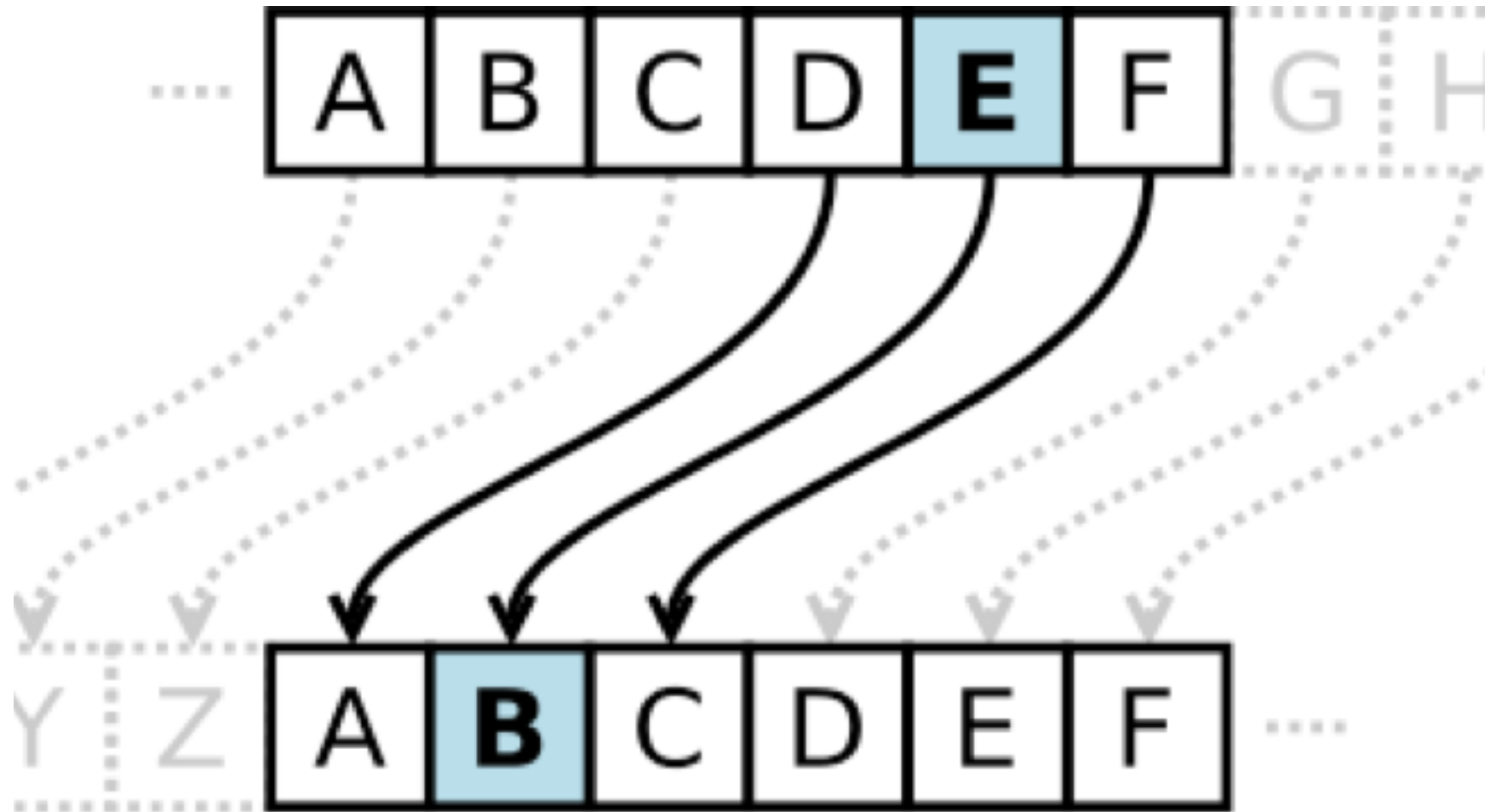
```
hi = 'Hello World'
```

```
hey = 'Hey world!'
```

```
print(hi.find('ello'), hey.find('ello'))
```

- A. 2 -1
- B. 1 -1
- C. 1 1
- D. 2 2
- E. Error

# Towards Caesar Cipher



---

# At the heart of the cipher is shifting letters

- Every letter is shifted  $k$  to left or right in alphabet, where combination of  $k$  and left or right is *encryption key*
- First **design decision**: Encode key (have inputs to shift function) as
  - 2 inputs: positive integer and "left" or "right"
  - 1 input, integer, with sign indicating left/right
    - And if 1 input, is negative shift left or right

---

# At the heart of the cipher is shifting letters

- Every letter is shifted  $k$  to left or right in alphabet, where combination of  $k$  and left or right is *encryption key*
- First **design decision**: Encode key (have inputs to shift function) as
  - 2 inputs: positive integer and "left" or "right"
  - 1 input, integer, with sign indicating left/right
    - And if 1 input, is negative shift left or right
  - All possible answers seem reasonable; so pick one and see how it works

# At the heart of the cipher is shifting letters

- Every letter is shifted  $k$  to left or right in alphabet, where combination of  $k$  and left or right is *encryption key*
- First **design decision**: Encode key (have inputs to shift function) as
  - 2 inputs: positive integer and "left" or "right"
  - 1 input, integer, with sign indicating left/right
    - And if 1 input, is negative shift left or right
  - Our arbitrary answer: + = shift right/up in alphabet:
    - + 2 means shift A to C

# Example: +3 Shift

ABCDEFGHIJKLMNOPQRSTUVWXYZ → +3 SHIFT  
DEFGHIJKLMNOPQRSTUVWXYZ???

What do we do when we run off the end?

Wrap around!

ABCDEFGHIJKLMNOPQRSTUVWXYZ → +3 SHIFT  
DEFGHIJKLMNOPQRSTUVWXYZABC

# How do we implement k shift?

- We need letter number  $i$  transformed to letter  $i + k$ , assuming  $i + k$  is not past  $z$  (i.e., 25), and to wrap otherwise
- First step: How do we find what letter number a given character is in the alphabet?
  - A. find
  - B. Indexing
  - C. String addition
  - D. Using the remainder function %

# Finding letter in alphabet

```
ALPHABET = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

```
>>> ALPHABET.find('A')
```

- A. 0
- B. 1
- C. Error
- D. Non-error other than 0 or 1



# find() gives position in alphabet

- Using Python 0 to 25 numbering!
- And adding k to go forward just works
  - *if* we don't fall off the end
- Trick you will use repeatedly as computer scientist: remainder (also called mod) operator % works for "wrapping around"
- E.g., Z forward 3 should give C
- $(25 + 3) \% 26 \rightarrow 2$ 
  - Recall ALPHABET[25]  $\rightarrow$  Z

# % for rotating, wrapping around

- Wikipedia and others describe Caesar cipher as using **shift** of letters
- Computer scientists often speak of **rotation**
- In your head, think of 26 letters of alphabet in a circle not in a line

# You have the pieces of rotate\_letter

```
def rotate_letter(character, k):  
    """Shifts character k positions  
    right for k>0, wrapping around if past z"""  
    ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
    ...
```

---

# But need some more Python to

- Handle left shifts
  - Julius Caesar's eventual successor, Augustus used a shift *left* of 1
  - And need opposite direction shift to decrypt
- *Encrypt whole string, such as "ATTACK" or "RETREAT" instead of just 1 character*
- *Handle mixed case and spaces, as in "ATTACK AT DAWN" and "Retreat now"*