

# STRING MANIPULATION, GUESS-and-CHECK, APPROXIMATIONS, BISECTION

(download slides and .py files ••• follow along!)

---

6.0001 LECTURE 3

# LAST TIME

---

- strings
- branching – if/elif/else
- while loops
- for loops

# TODAY

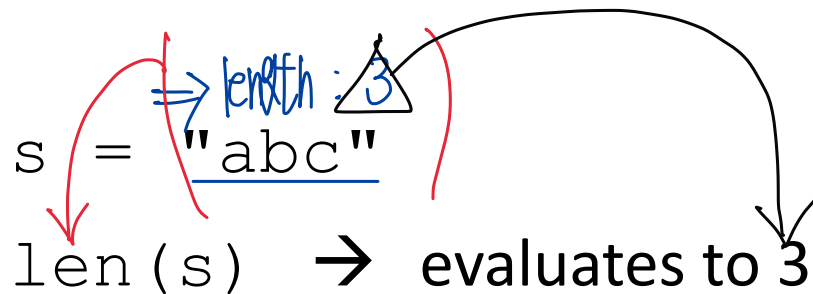
---

- string manipulation
- guess and check algorithms
- approximate solutions
- bisection method

# STRINGS

---

- think of as a **sequence** of case sensitive characters
- can compare strings with `==`, `>`, `<` etc.
- `len()` is a function used to retrieve the **length** of the string in the parentheses



# STRINGS

- square brackets used to perform **indexing** into a string to get the value at a certain index/position

s = "abc"

index: 0 1 2 ← indexing always starts at 0

index: -3 -2 -1 ← last element always at index -1

s[0] → evaluates to "a"

s[1] → evaluates to "b"

s[2] → evaluates to "c"

~~s[3] → trying to index out of bounds, error~~

s[-1] → evaluates to "c"

s[-2] → evaluates to "b"

s[-3] → evaluates to "a"

	P(+)	N(-)
a	0	-3
b	1	-2
c	2	-1

max  
position

: 2

\*\*\*

# STRINGS

- can **slice** strings using `[start:stop:step]`
- if give two numbers, `[start:stop]`, `step=1` by default
- you can also omit numbers and leave just colons

`s = "abcdefgh"`  
0 1 2 3 4 5 6 7

`s[3:6]` → evaluates to "def", same as `s[3:6:1]`

`s[3:6:2]` → evaluates to "df" 

`s[::]` → evaluates to "abcdefgh", same as `s[0:len(s):1]`

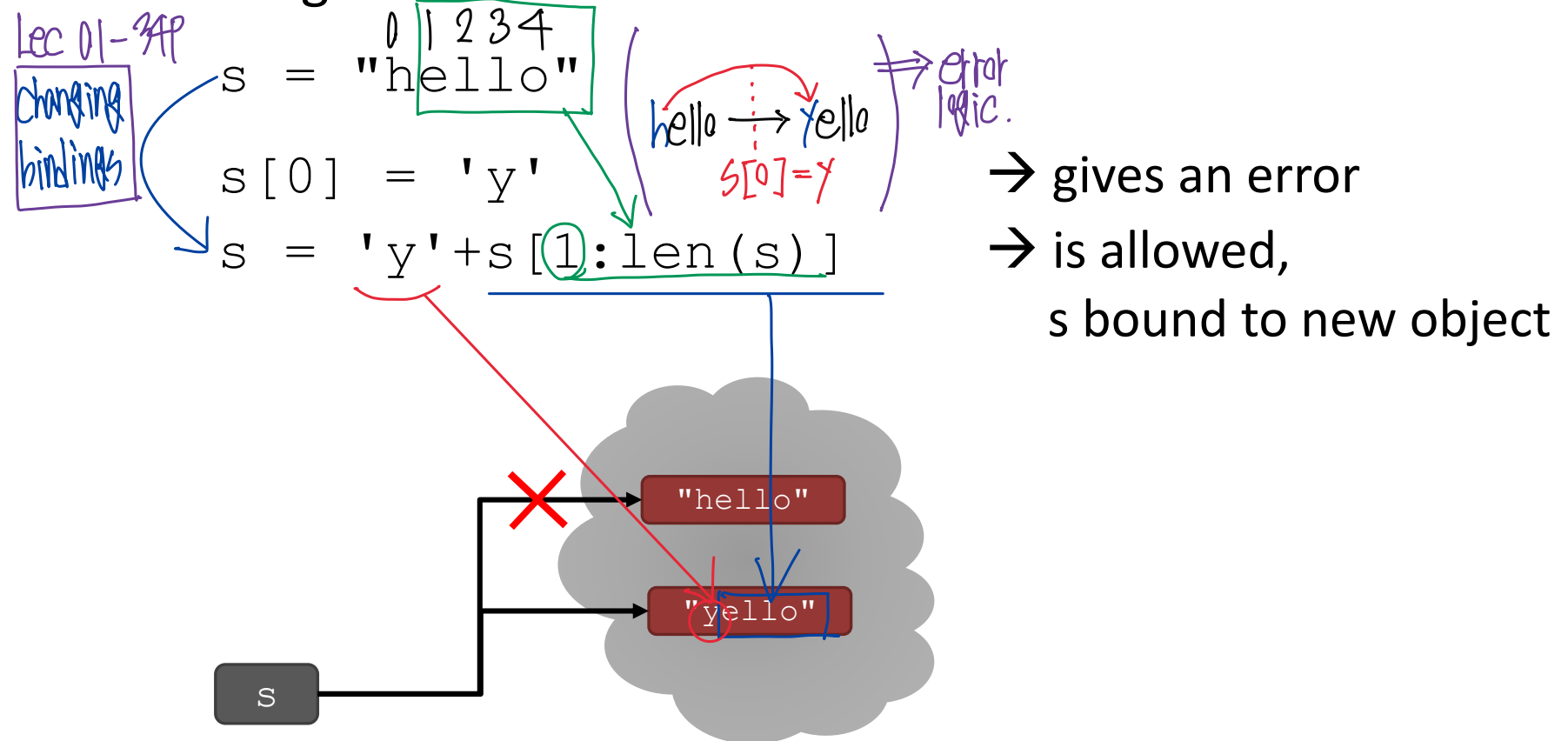
`s[::-1]` → evaluates to "hgfedbca", same as `s[-1:- (len(s)+1) : -1]`

`s[4:1:-2]` → evaluates to "ec" 

*If unsure what some command does, try it out in your console!*

# STRINGS

- strings are “**immutable**” – cannot be modified



# for LOOPS RECAP

---

- for loops have a **loop variable** that iterates over a set of values

for var in <sup>⇒ 0,1,2,3</sup>range(4) :      → var iterates over values 0,1,2,3  
    <expressions>      → expressions inside loop executed  
                                with each value for var


for var in range(4, 6) :      → var iterates over values 4,5  
    <expressions>

- range is a way to iterate over numbers, but a for loop variable can **iterate over any set of values**, not just numbers!



# STRINGS AND LOOPS

---

- these two code snippets do the same thing
- bottom one is more “pythonic”

```
s = "abcdefgh"
```

```
for index in range(len(s)):
```

```
    if s[index] == 'i' or s[index] == 'u':
```

```
        print("There is an i or u")
```

```
for char in s:
```

```
    if char == 'i' or char == 'u':
```

```
        print("There is an i or u")
```

# CODE EXAMPLE: ROBOT CHEERLEADERS

---

```
an_letters = "aefhilmnorsxAEFHILMNORSX"
```

```
word = input("I will cheer for you! Enter a word: ")  
times = int(input("Enthusiasm level (1-10): "))
```

```
i = 0  
while i < len(word):  
    char = word[i]  
    if char in an_letters:  
        print("Give me an " + char + "! " + char)  
    else:  
        print("Give me a  " + char + "! " + char)  
    i += 1  
print("What does that spell?")  
for i in range(times):  
    print(word, "!!!")
```

for char in word:



# EXERCISE

---

```
s1 = "mit u rock"
s2 = "i rule mit"
if len(s1) == len(s2):
    for char1 in s1:
        for char2 in s2:
            if char1 == char2:
                print("common letter")
                break
```

# GUESS-AND-CHECK

---

- the process below also called **exhaustive enumeration**
- given a problem...
- you are able to **guess a value** for solution
- you are able to **check if the solution is correct**
- keep guessing until find solution or guessed all values

# GUESS-AND-CHECK

## – cube root

---

```
cube = 8
for guess in range(cube+1):
    if guess**3 == cube:
        print("Cube root of", cube, "is", guess)
```

# GUESS-AND-CHECK

## – cube root

---

```
cube = 8
```

```
for guess in range(abs(cube)+1):
```

```
    if guess**3 >= abs(cube):
```

```
        break
```

```
if guess**3 != abs(cube):
```

```
    print(cube, 'is not a perfect cube')
```

```
else:
```

```
    if cube < 0:
```

```
        guess = -guess
```

```
    print('Cube root of ' + str(cube) + ' is ' + str(guess))
```

~~\*\*\*~~

⇒ for math.

cf)  $guess = -guess$  ~~≠~~  $guess + (-guess) = 0$   
Computer can understand      Computer can't understand.

# APPROXIMATE SOLUTIONS

---

- **good enough** solution
- start with a guess and increment by some **small value**
- keep guessing if  $| \text{guess}^3 - \text{cube} | \geq \text{epsilon}$   
for some **small epsilon**  

$\Rightarrow$  in Python ) `abs( )`  
comment  
`if abs(Luciano) : | Luciano |`
- decreasing increment size  $\rightarrow$  slower program
- increasing epsilon  $\rightarrow$  less accurate answer

# APPROXIMATE SOLUTION

## – cube root

---

```
cube = 27
epsilon = 0.01
guess = 0.0
increment = 0.0001
num_guesses = 0
while abs(guess**3 - cube) >= epsilon and guess <= cube :
    guess += increment
    num_guesses += 1
print('num_guesses =', num_guesses)
if abs(guess**3 - cube) >= epsilon:
    print('Failed on cube root of', cube)
else:
    print(guess, 'is close to the cube root of', cube)
```



cf)

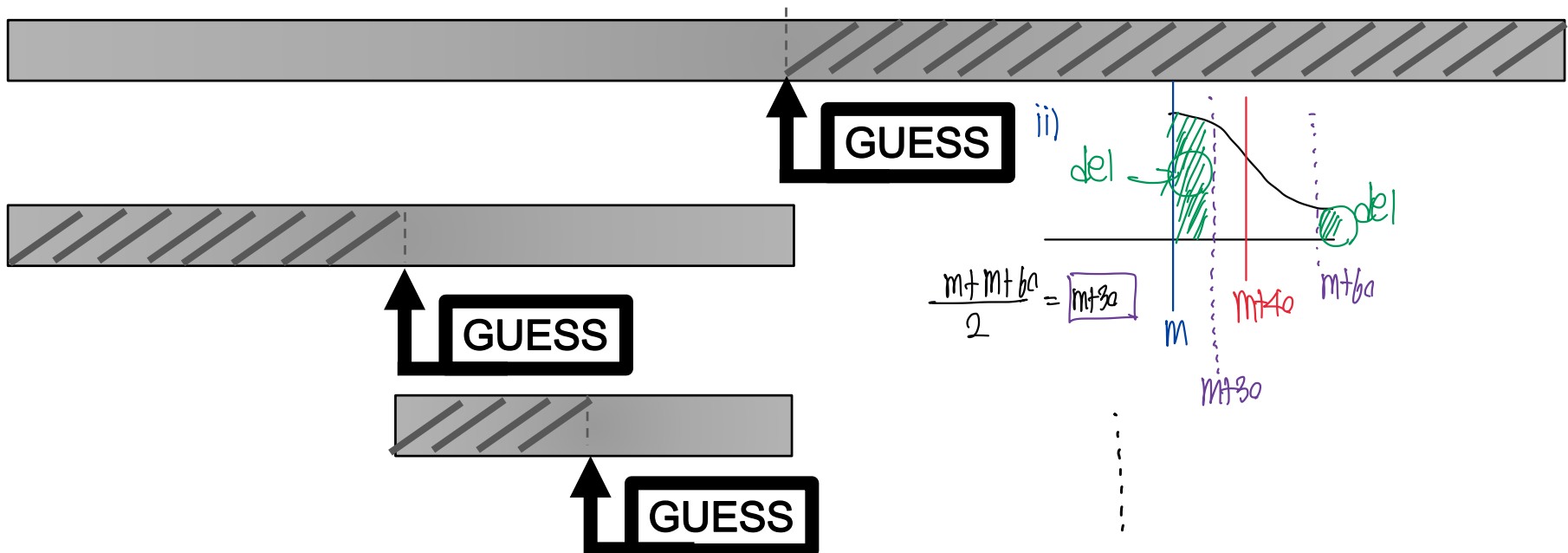
i) We want

$m+4\sigma$

$m+12\sigma$

$$\frac{m + m+12\sigma}{2} = m+6\sigma$$

- 



# BISECTION SEARCH

## – cube root

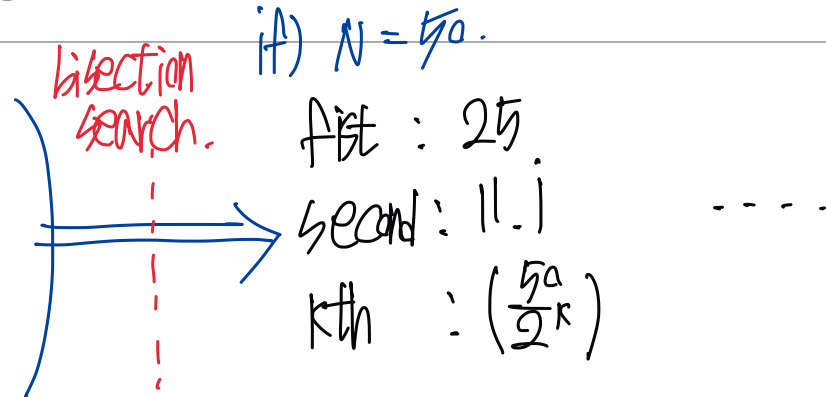
---

```
cube = 27
epsilon = 0.01
num_guesses = 0
start Point low = 0
high = cube
guess = (high + low)/2.0
while abs(guess**3 - cube) >= epsilon:
    if guess**3 < cube :
        low = guess
    else:
        high = guess
    guess = (high + low)/2.0
    num_guesses += 1
print 'num_guesses =', num_guesses
print guess, 'is close to the cube root of', cube
```

# BISECTION SEARCH CONVERGENCE

- search space

- first guess:  $N/2$
- second guess:  $N/4$
- kth guess:  $N/2^k$



- guess converges on the order of  $\log_2 N$  steps

- bisection search works when value of function varies monotonically with input

if  $N = 1000$ : Try 10 guesses ↓

T:	1	2	3	4	...	Always increase
F:	1	2	4	3	-1	... ~ Always

- code as shown only works for positive cubes  $> 1$  – why?

- challenges
  - modify to work with negative cubes!
  - modify to work with  $x < 1$ !

$$x < 1$$

---

- if  $x < 1$ , search space is 0 to  $x$  but cube root is greater than  $x$  and less than 1
- modify the code to choose the search space depending on value of  $x$

MIT OpenCourseWare

<https://ocw.mit.edu>

6.0001 Introduction to Computer Science and Programming in Python

Fall 2016

For information about citing these materials or our Terms of Use, visit: <https://ocw.mit.edu/terms>.