Topic 3

Strings and String Manipulation

Murdoch
UNIVERSITY

LAST WEEK

Murdoch UNIVERSITY

- Data
- Data Types
- Objects
- Variables
- Simple operations
- Expressions
- Assignment statement
- Output statement `print( … )`
- Branching and conditionals
- Indentation
- Iteration and loops

TODAY

Murdoch UNIVERSITY

- Define multi-line string
- String index: positive index and negative index
- Slicing a string
- Strings are immutable, string identity
- Loop over the sequence of characters in a string,
- Formatted strings
- Strings as composite objects
- String methods
- F-strings
- Escape character

MULTI-LINE STRINGS

Murdoch UNIVERSITY

- We can define a string using single quotes or double quotes, but those strings must be in one line.
- Using three quotes, either single or double, we can define a string that crosses multiple lines, eg:

```
s1 = '''The way to get started is
to quit talking and begin doing … '''
s2 = """The word "Python" doesn't always
mean programming language"""
```

- Note: a triple quoted string may contain single quote and double quote characters. Eg,

```
sentence = '''"It's ok", he said'''
```

CONVERT A STRING TO A NUMBER

Murdoch UNIVERSITY

- We often get a number in the string format, eg, with the `input` function.
- These string-like numbers, such as `"123"`, `"56.7"`, etc cannot participate in numerical calculations.

```
s = "123"
sum = s + 3  # produce a type error
```

- We must convert these string-like numbers into ints or floats before the calculation:

```
s = "123.5"
n1 = int(s)    # return 123
n2 = float(s)    # return 123.5
n3 = int("123x")  # produce a value error
```

STRING LENGTH

Murdoch UNIVERSITY

- think of a string 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

```
s = "abc"

len(s)  # evaluates to 3
```

STRING INDEX

Murdoch UNIVERSITY

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

```
s = "abc"
```

positive index: 0, 1, 2 => indexing always starts at 0
Negative 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"
s[-4]       # index error: string index out of range
```

STRING SLICING

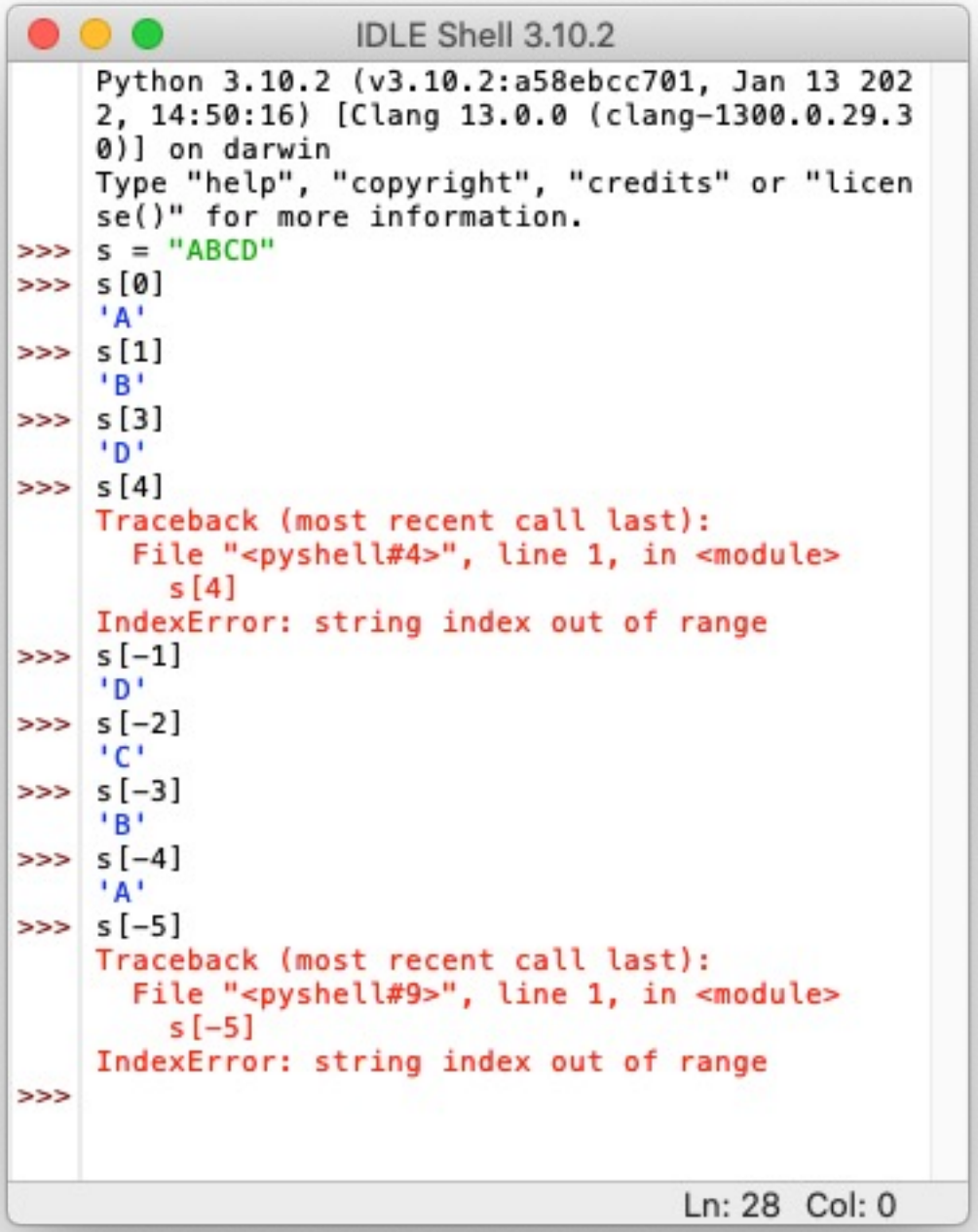
Murdoch UNIVERSITY

- you can **slice** strings using `[start:stop:step]`
  - Note the syntax for indexing a string is the same as the range function
- with two numbers, [`start:stop`], `step=1` by default
- you can also omit numbers and leave just colons

```
s = "abcdefgh"
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 "hgfedcba", same as s[-1:-(len(s)+1):-1]
```

STRING SLICING

Murdoch UNIVERSITY

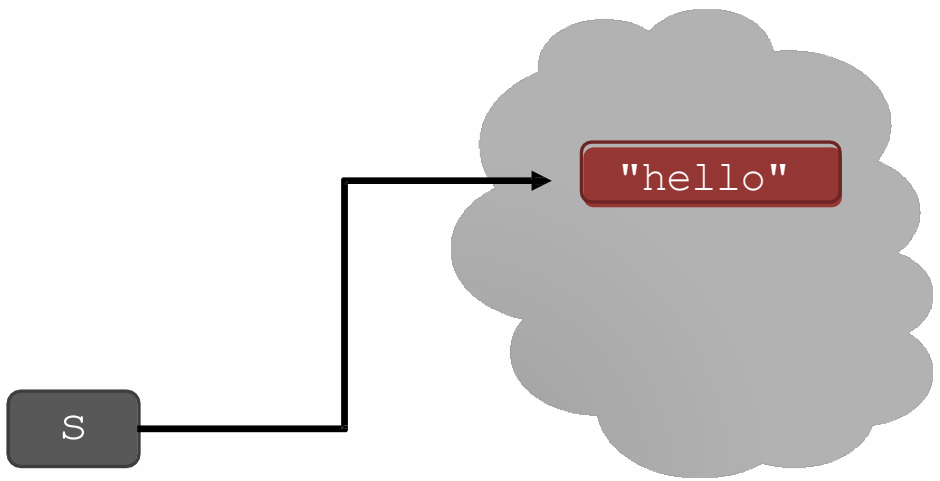


STRINGS ARE IMMUTABLE

Murdoch UNIVERSITY

- strings are "**immutable**" – cannot be modified

```
s = "hello"
s[0] = 'y'    # gives an error
```



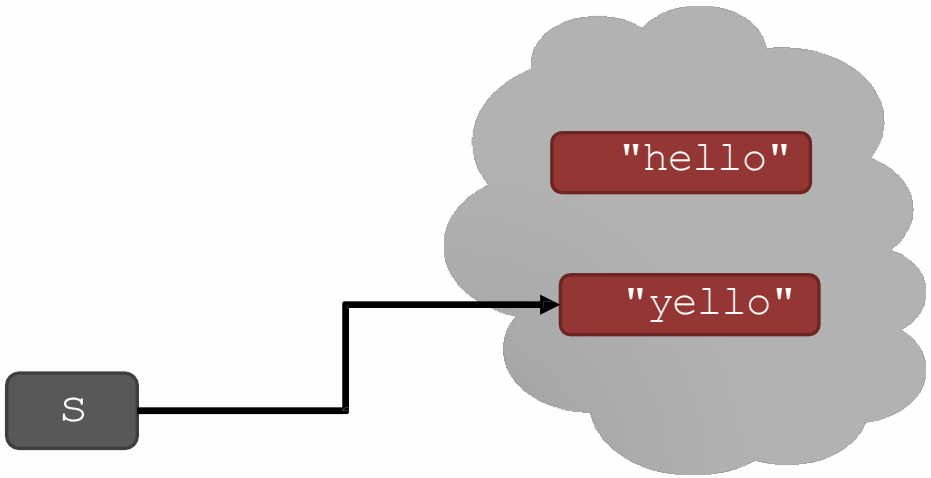
STRINGS ARE IMMUTBLE

Murdoch UNIVERSITY

- strings are "**immutable**" – cannot be modified

```
s = "hello"
s[0] = 'y'                  # gives an error
s = 'y'+s[1:len(s)]         # is allowed
```

In this case, `s` is bound to a new object (s is assigned a new address pointing to a new string object "yello").



STRING IDENTITY

Murdoch UNIVERSITY

- In the example in the previous slide, variable `s` points to string `"hello"` first and then to string `"yello"`.

- Each string has an id. To see these strings are actually different strings, we can print their ids:

```
s = "hello"
print(id(s))     # 4385598000
s = 'y'+s[1:len(s)]
print(id(s))     # 4354671472
```

Note: these ids are dynamically generated, so you may see different ids when you try the above code.

CHECK SUBSTRING WITH in OPERATOR

Murdoch UNIVERSITY

- The operator in can be used to check whether a string is a substring in another string. Eg

```
>>> ict582 = "Python Programming"
>>> print("ram" in ict582)
True
>>> print("ran" in ict582)
False
>>>
```

- An empty string is always a substring of another string. Eg

```
>>> '' in ict582
True
>>>
```

Note: `x in s` is equivalent to `s.find(x) != -1`

for LOOPS RECAP

Murdoch UNIVERSITY

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

```
for var in range(4):
    <statement>
```

  - `var` iterates over values 0,1,2,3. The statement inside the loop executes with each value for `var`

```
for var in range(4,6):
    <statement>
```

  - `var` iterates over values 4,5.

- `range` function provides a way to iterate over a sequence of numbers, but a `for` loop variable can **iterate over any sequence of values**, not just numbers!

STRING AS A SEQUENCE OF CHARACTERS

Murdoch UNIVERSITY

- As a string can be considered as a sequence of characters, these two code snippets do the same thing
- The bottom one is more “pythonic”

```
s = "abcdefghijk"                    # 11 characters

for index in range(len(s)): # index = 0, 1, 2, . ., 10
    if s[index] == 'i' or s[index] == 'u':
        print("There is an i or u")

for char in s:  # char = 'a', 'b', 'c', . . . , 'k'
    if char == 'i' or char == 'u':
        print("There is an i or u")
```

EXAMPLE: ROBOT CHEERLEADERS

Murdoch UNIVERSITY

```
# letters of words that require article "an":
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   # same as i = i + 1
print("What does that spell?")
for i in range(times):
    print(word, "!!!")
```

EXAMPE:
ROBOT CHEERLEADERS

Murdoch UNIVERSITY

```
C:\Users\Hong\ict582\t03\tests>python test1.py
I will cheer for you! Enter a word: wonderful
Enthusiasm level (1-10): 3
Give me a w! w
Give me an o! o
Give me an n! n
Give me a d! d
Give me an e! e
Give me an r! r
Give me an f! f
Give me a u! u
Give me an l! l
What does that spell?
wonderful !!!
wonderful !!!
wonderful !!!
```

EXAMPLE: ROBOT CHEERLEADERS

Murdoch UNIVERSITY

```
# letters of words that require article "an":
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

Murdoch UNIVERSITY

```
s1 = "MU u rock"
s2 = "i rule MU"

# find the common characters
# in s1 and s2
for char1 in s1:
    for char2 in s2:
        if char1 == char2:
            print("a common character: " + char1)
            break
```

STRING FORMATTING

Murdoch UNIVERSITY

- We often need to create a complex string containing values of different variables, such as integers, floats and strings.
- We also need to control the output formats of such numbers, eg, number of decimal places after the decimal point.

```
coffee = "flat white"
cost = 5.35
s = "You ordered %s and the cost is %0.2f" %(coffee, cost)
print(s)
```

- Note: in the above, `%s` is replaced by the value of the first variable (`coffee`) and `%0.2f` is replaced by the value in the second variable (`cost`).
  - The format `%s` means a string is expected here
  - The format `%0.2f` means it's a float accurate to 2nd decimal place.

STRING FORMATTING

Murdoch UNIVERSITY

- If there is only one string substitution, no need to have parentheses after `%` operator.

```
temerature = 38.5
print("you have a fever: %f" %temperature)
```

  - Note: in the above, format `%f` is substituted by the value in variable `temperature`, with the precision of six decimal places after the decimal point!

```
>>>
>>> temperature = 38.5
...
>>> print("you have a fever: %f" %temperature)
...
you have a fever: 38.500000
>>>
```

STRING FORMATTING

Murdoch UNIVERSITY

- The character % has a special meaning in a formatted string – it indicates a format and what to be replaced by the value of a variable.
- What should we do if we need to use % character in a formatted string, eg, to denote a percentage?

```
inflation = 0.085
s = "rent has increased %0.1f% this year" % (inflation*100)
```

- The above statement will cause a TypeError. To correct it, we need to use %% for percentage character in a formatted string:

```
s = "rent has increased %0.1f%% this year" % (inflation*100)
```

```
>>> print(s)
rent has increased 8.5% this year
>>>
```

A STRING IS AN OBJECT!

Murdoch UNIVERSITY

- In Topic 1 and Topic 2, we learnt that an integer, a float or a Boolean is a scalar object.
- This implies that some other values are non-scalar objects. A string happens to be a non-scalar or composite object!
- You can think of a composite object as a wrap or container inside which there are many pieces of data values
- Apart from the data, an object also contains many methods (just like functions, we call them “methods” if they are defined inside a class, more on this in Topic 9)
- An object has two components:
  - a state which is defined by the current data values
  - a set of behaviours defined by the methods of the object that can operate on these data values

A STRING IS AN OBJECT!

Murdoch UNIVERSITY

- For example, the string `"abc"` not only contains the character sequence `a`, `b`, `c`, but also a list of methods that can do something on `"abc"`. Eg, the method `upper()` returns the string in upper case:

```
print("abc".upper()) # output ABC
```

- In fact, notations such as `s[2]` is an invocation of a method to retrieve the 3rd character from the string `s`.
  - You can think of it as a shorthand for `s.chartAt(2)`.
- As strings are used so frequently in programming, Python provided many special syntax to simplify the use of these string methods.

STRING METHODS

Murdoch UNIVERSITY

- A string object contains many built-in methods which can operate on the string. To see what methods are available inside a string , you can use function `dir` on the string:

```
s = "Python is fun!"

print(dir(s))
```

```
>>> s = "Python is fun!"
>>> print( dir(s) )
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '_
_eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs
__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__'
, '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__'
, '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'e
ncode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isal
num', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', '
isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lo
wer', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix', 'repla
ce', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'spl
itlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfi
ll']
>>>
```

STRING METHODS

Murdoch UNIVERSITY

- You can ignore those methods that start and end with a double underscore, such as `__add__` and `__class__`. They are rarely used in normal Python programming
- If you do not know how to use a particular method, use the function help:

```
>>> s = "abc"
...
>>> print(help(s.capitalize))
...
Help on built-in function capitalize:

capitalize() method of builtins.str instance
    Return a capitalized version of the string.

    More specifically, make the first character have upper case and the rest low
er
    case.

None
>>>
```

STRING METHODS

Murdoch UNIVERSITY

- The following are some of the commonly used string methods:
  - `count(`*`string`*`)` : returns the number of times a specified value occurs in the string

```
s = "  Our python is not your python!"
print(s.count("python"))  # print 2
```

  - `find(`*`string`*`)` : searches the string with a specific value and returns the position of where it was found

```
print(s.find("python"))  # print 7
```

  - `isalpha()`: returns True if all characters in the string are letters

```
print(s.isalpha())  # print False (because !)
```

  - `lstrip()`: returns a left trim version of the string

```
print(s.lstrip()) # remove the two spaces at the beginning
```

STRING METHODS

Murdoch UNIVERSITY

- For more string methods and their examples, see the following tutorial:
- https://www.w3schools.com/python/python_ref_string.asp

STRING METHOD `format`

Murdoch UNIVERSITY

- `format(a, b, c)`: replace the *numbered placeholders* in the string with a, b, and c

```
>>> name = "John"
>>> classId = "TU08NH"
>>> s = "Hello {0}, your class id is {1}".format(name, classId)
>>> s
'Hello John, your class id is TU08NH'
>>>
```

- A placeholder may contain a format specifier. Eg

```
>>> price = 12.45
>>> number = 15
>>> s = "Total cost is {0:.2f}".format(price*number)
>>> s
'Total cost is 186.75'
>>>
```

F-STRINGS

Murdoch UNIVERSITY

- Similar to string method format, one can create a new string by replacing the *named placeholders with* the values of the respective variables

```
>>> name = "John"
>>> classId = "TU08NH"
>>> s = f"Hello {name}, your class id is {classId}"
>>> print(s)
Hello John, your class id is TU08NH
>>>
```

- Note that the string must start with character `f`, hence this types of strings are also called *f-strings*.

F-STRINGS

Murdoch UNIVERSITY

- In a f-string, the placeholder may contain an expression rather than a variable name. Eg

```
>>> s = f"A year has {24*365} hours"
>>> s
'A year has 8760 hours'
>>>
```

- In general, a placeholder in a f-string may contain any legitimate expression. Eg

```
>>> code = "ICT582"
>>> unit = "Python Programming Principles and Practice"
>>> s = f"The title of unit {code} has {len(unit)} characters"
>>> s
'The title of unit ICT582 has 42 characters'
>>>
```

F-STRING

Murdoch UNIVERSITY

- Like the string method `format`, in a f-string, a placeholder may contain a format specifier. Eg

```
>>> price = 12.45
>>> number = 15
>>> s = f"Total cost is {price*number:.2f}"
>>> print(s)
Total cost is 186.75
>>>
```

ESCAPE CHARACTER \

Murdoch UNIVERSITY

- Some characters have special meanings in a string, such as single quote ' and double quote ". For these special characters to be appear in a string as a normal character, you need to escape its special meaning. Eg

```
>>>
>>> s = "\"Good morning\", he said"
>>> s
'"Good morning", he said'
>>>
```

- Other special characters, eg

```
\', \\, \t, \b, \n, \r
```

Murdoch
UNIVERSITY

Algorithms

This part is optional.

GUESS-AND-CHECK

Murdoch UNIVERSITY

- 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

Murdoch UNIVERSITY

```
cube = 8
# assume the cubic root is between 0 and cube
for guess in range(cube+1):
    if guess**3 == cube:
        print("Cube root of", cube, "is", guess)
```

GUESS-AND-CHECK – cube root

Murdoch UNIVERSITY

```
cube = int(input("Enter an integer: "))
for guess in range(cube+1):
    if guess**3 >= cube:
        break
if guess**3 != cube:
    print(cube, 'is not a perfect cube')
else:
    print('Cubic root is', guess)
```

Question: can this program handle negative number?

APPROXIMATE SOLUTIONS

Murdoch UNIVERSITY

- **good enough** solution
- start with a guess and increment by some **small value**
- keep guessing if `|guess^3-cube| >= epsilon` for some **small epsilon**
- decreasing increment size -> slower program
- increasing epsilon -> less accurate answer

APPROXIMATE SOLUTION – cube root

Murdoch UNIVERSITY

```
cube = 26
epsilon = 0.01
guess = 0.0
increment = 0.0001
num_guesses = 0
while abs(guess**3 - cube) >= epsilon:
    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)
```

APPROXIMATE SOLUTION – cube root

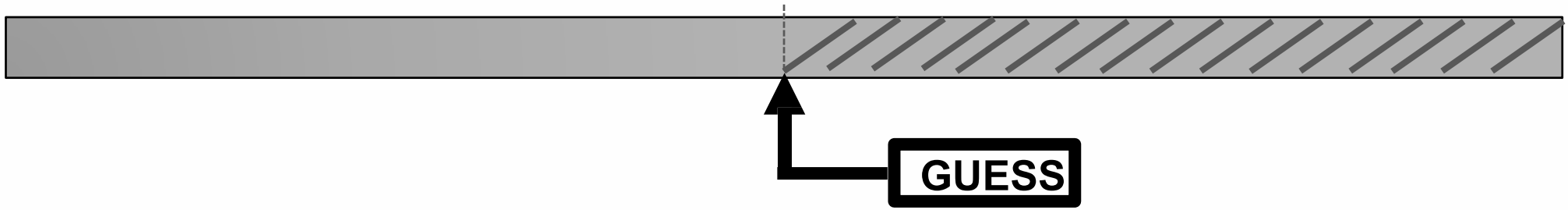
Murdoch UNIVERSITY

```
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)
```

BISECTION SEARCH

Murdoch UNIVERSITY

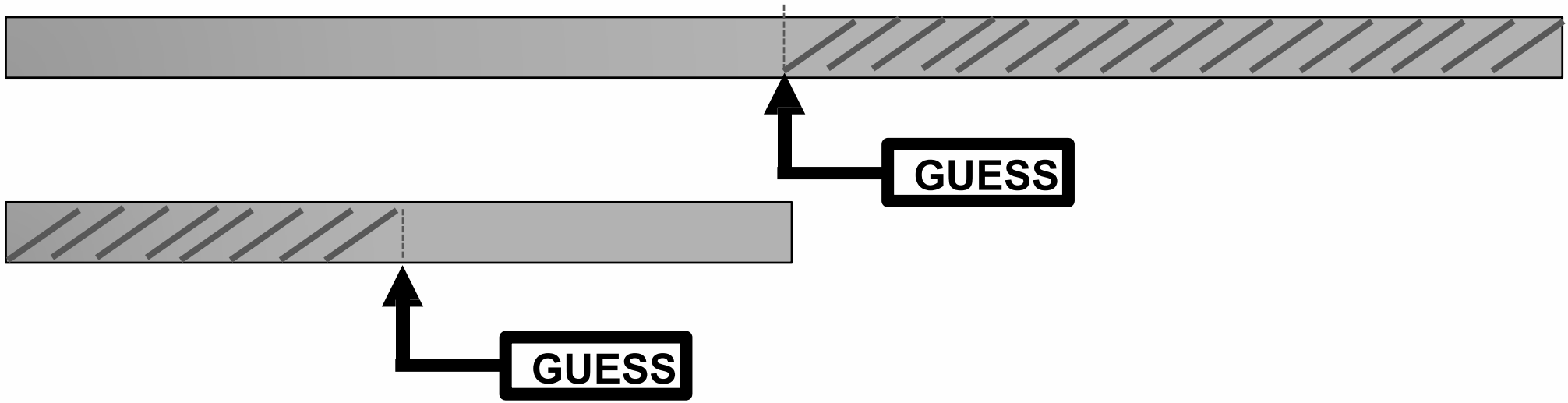
- half interval each iteration
- new guess is halfway in between
- to illustrate, let's play a game!



BISECTION SEARCH

Murdoch UNIVERSITY

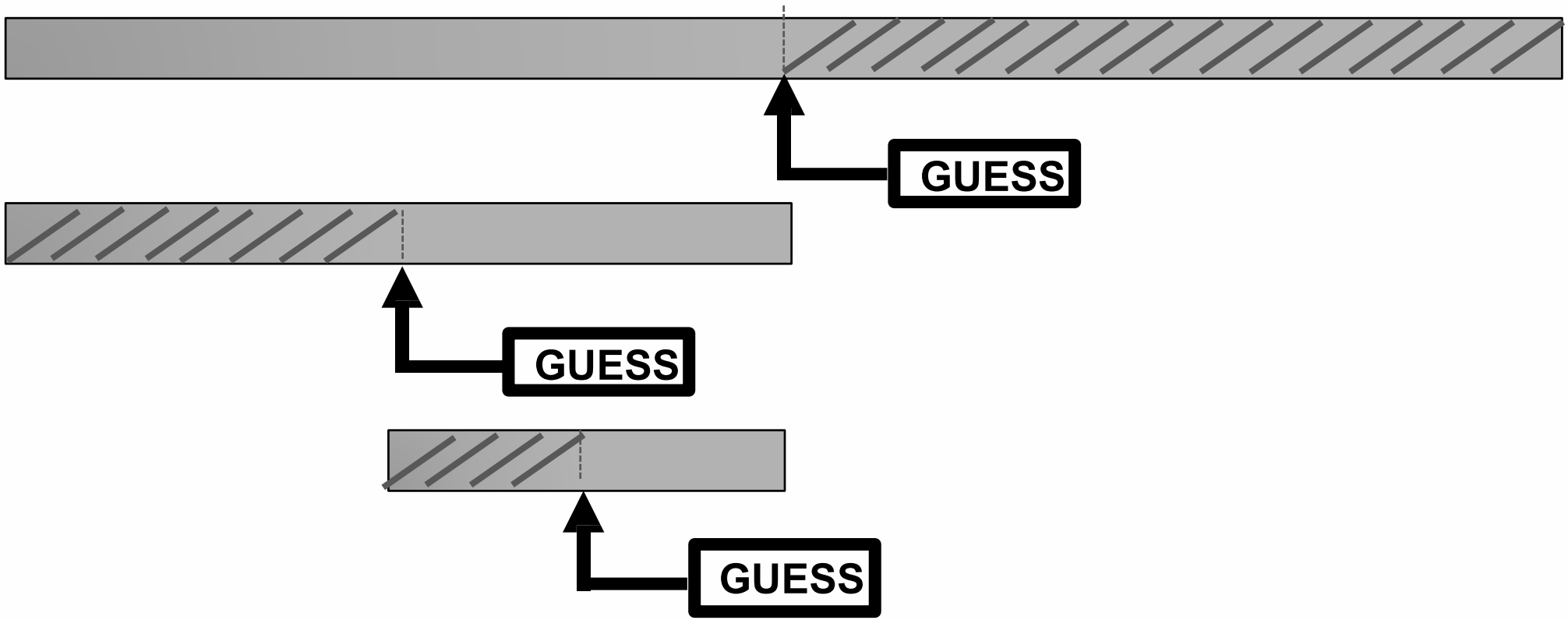
- half interval each iteration
- new guess is halfway in between
- to illustrate, let's play a game!



BISECTION SEARCH

Murdoch UNIVERSITY

- half interval each iteration
- new guess is halfway in between
- to illustrate, let's play a game!



BISECTION SEARCH – cube root

Murdoch UNIVERSITY

```
cube = 27
epsilon = 0.01
num_guesses = 0
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

Murdoch UNIVERSITY

- 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
- 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 (Exercise)

Murdoch UNIVERSITY

- 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

Acknowledgement/Useful Resources

Murdoch UNIVERSITY

- Sources used in this presentation include:
  - Programiz.com
  - MIT OCW
- Useful links
  - Pythontutor.com
  - https://www.w3schools.com/python/