Strings

Words!

Housekeeping

- Fill out midquarter feedback form!
- Midterm is Wednesday, July 26th
- Content covers **through yesterday's lecture**
- Breakout due Sunday
- Tea time after class today!

Today

- **Python main + arguments**
  - **Recap how we process input from the command line**

- Introduce Strings
  - String datatype
  - How to process text in Python
  - Strings as parameters

Arguments

- When I run **`python3 example.py Frankie`**, the string “Frankie” becomes an “argument”
- Arguments don’t do much until we tell main to use them

example.py

```
def func():
  print("first line of func!")

def main():
  print("first line of pgm!")
  my_helper()
```

Terminal:

```
$ python3 example.py Frankie
first line of pgm!
first line of func!
```

Arguments

- We can access a **list** of arguments like so

example.py

```
import sys

def main():
  args = sys.argv[1:]

  print(args)
```

Terminal:

```
$ python3 example.py Frankie 106A DVDs
```

[Frankie, 106A, DVDs]

Recall: List slicing

- sys.argv includes the file name too, which we “slice” off

example.py

```
import sys

def main():
  all_args = sys.argv
  #all_args:[example.py, Frankie, 106A, DVDs]
  args = all_args[1:]
  #args:[Frankie, 106A, DVDs]
```

Terminal:

```
$ python3 example.py Frankie 106A DVDs
```

Recall: List slicing

- In general, **`list[start:end]`** will take the "slice" of the list starting at index start and ending before index end
- If we omit **`start`** or **`end`**, it will treat it as **`0`** or **`len(list)`**, respectively

```
def main():
  list = [5, 6, 7, 8]
  a = list[1:3] # [6, 7]
  b = list[:3]  # [5, 6, 7]
  c = list[0:]  # [5, 6, 7, 8]
```

Arguments

- By default, all arguments are interpreted as strings

example.py

```
import sys

def main():
  args = sys.argv[1:]

  print(args[0] + 1)
```

Terminal:

```
$ python3 example.py 10
```

???

Arguments

- By default, all arguments are interpreted as strings

example.py

```
import sys

def main():
  args = sys.argv[1:]

  print(args[0] + 1)
```

Terminal:

```
$ python3 example.py 10
```

Error: Can't add int and string

Arguments

- By default, all arguments are interpreted as strings
- To interpret arguments as another type, transform them with **`type(args[i])`**

example.py

```
import sys

def main():
  args = sys.argv[1:]

  print(int(args[0]) + 1)
```

Terminal:

```
$ python3 example.py 10
11
```

Arguments

- By default, a space specifies a new argument

```
import sys

def main():
  args = sys.argv[1:]

  print(args)
  print(len(args))
```

Terminal:

```
$ python3 example.py Frankie loves 106A
```

[Frankie, loves, 106A]

3

Arguments

- By default, a space specifies a new argument
- Make a multiword argument by using quotes in the command line

```
import sys

def main():
  args = sys.argv[1:]

  print(args)
  print(len(args))
```

Terminal:

```
$ python3 example.py "Frankie loves 106A"
```

[Frankie loves 106A]

1

mad_libs

- Write a program that takes in two numbers, a name, and an animal name as command line input

- Fill in the mad libs sentence below with the appropriate command line arguments and print:

NAME has NUM1*NUM2 ANIMALs

```
python3 mad_libs 3 2 Frankie dog
Frankie has 6 dogs
```

Today

- ~~Python main + arguments~~
  - ~~Recap how we process input from the command line~~

- **Introduce Strings**
  - **String datatype**
  - **How to process text in Python**
  - **Strings as parameters**

Q: How is text represented in Python?

A: Strings!

The variable type `string`

Text is stored using the variable type

`string`.

A **`string`** is a sequence of characters.

```
def main():
    text = 'hello!'
    print(text)
```

Terminal:

hello!

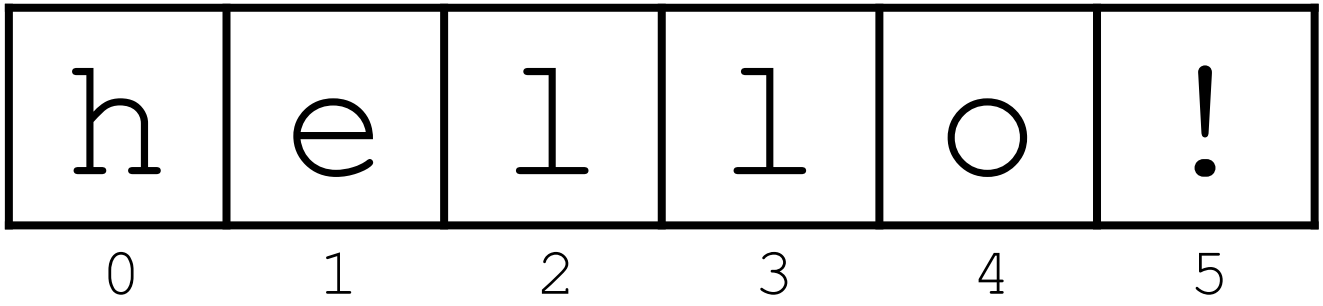
How are `strings` represented?

```
def main():
    text = 'hello!'
    print(text)
```

h e l l o !

How are `strings` represented?

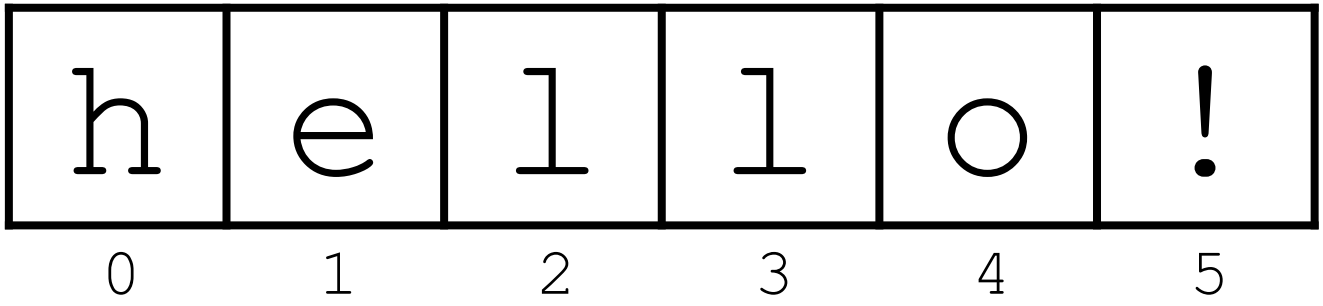
```
def main():
    text = 'hello!'
    print(text)
```



Indexed starting from 0

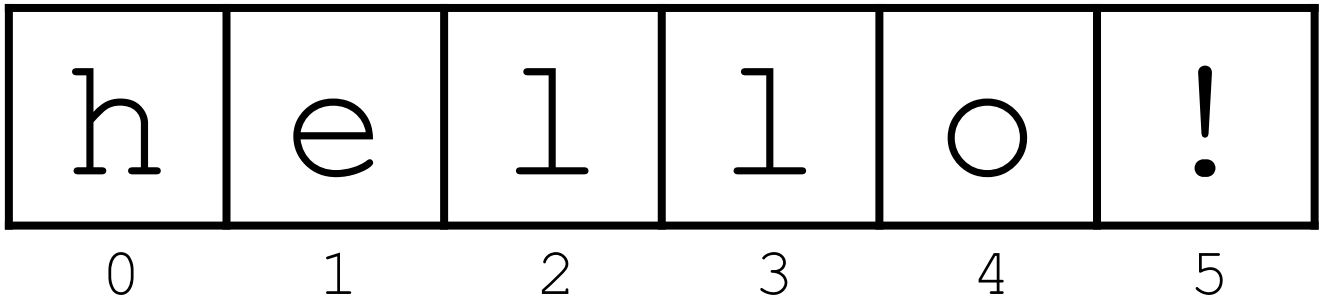
How are `strings` represented?

```
def main():
    text = 'hello!'
    print(text)
```




```
letter = text[0]  # letter = 0
```

How are `strings` represented?

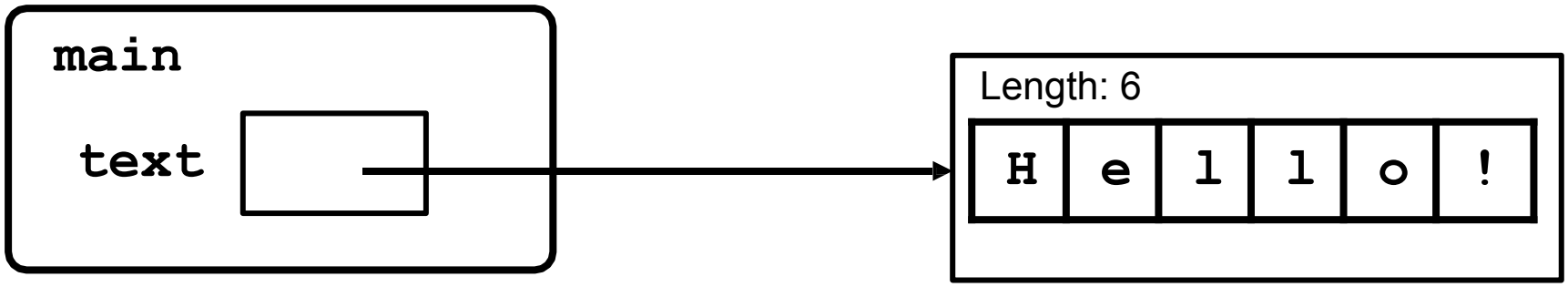


All characters in a string have an index.

You can access a character in the string via its *index*.

How it is stored

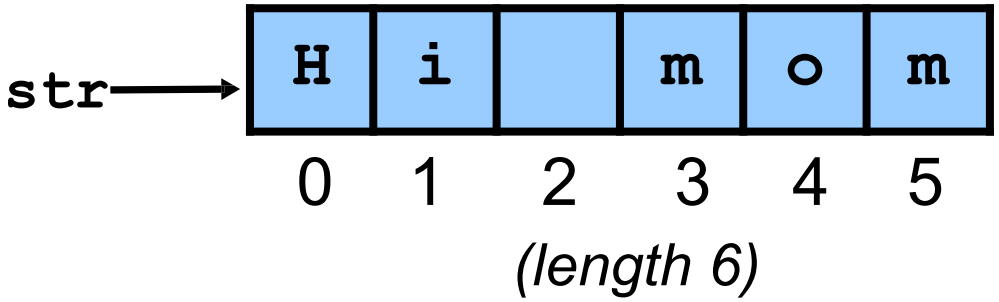
```
def main():
    text = 'hello!'
    print(text)
```



String Functions I

- Function: **`len`** (***`string`***)
  - Returns the number of characters in the string
- Function: ***`string`***`[`***`index`***`]`
  - Returns the character at the given index
  - A character is really just a string of length 1

```
def main():
    str = 'Hi mom'
    print(str[0])
    print(str[3])
    print(len(str))
```



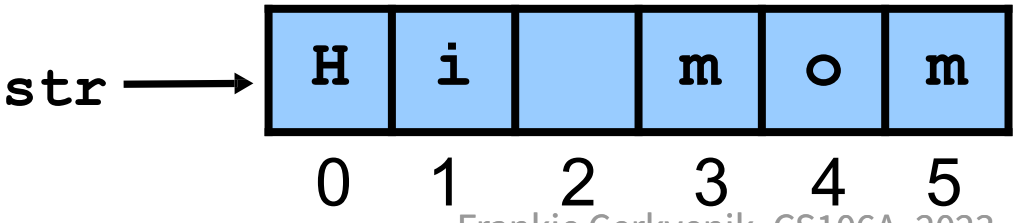
Working with Strings

```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Working with Strings

```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Terminal

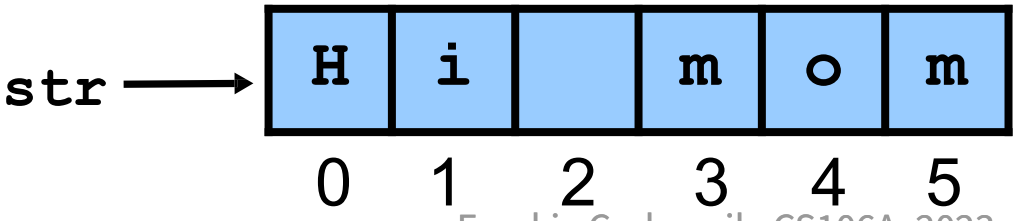


Working with Strings

```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Terminal

```
H
```

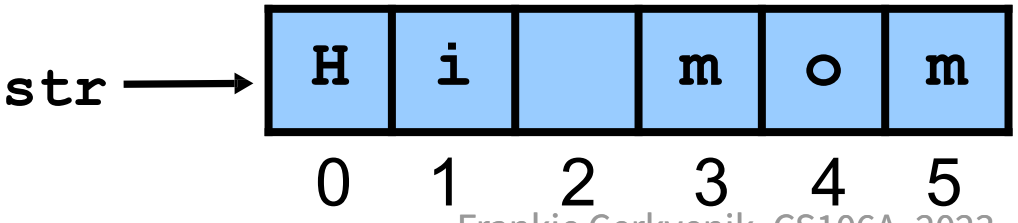


Working with Strings

```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Terminal

```
H
m
```

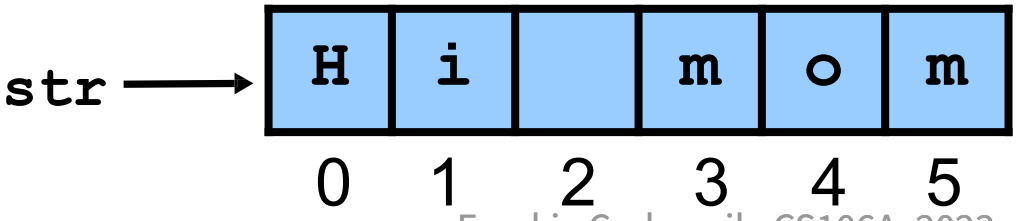


Working with Strings

```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Terminal

```
H
m
6
```

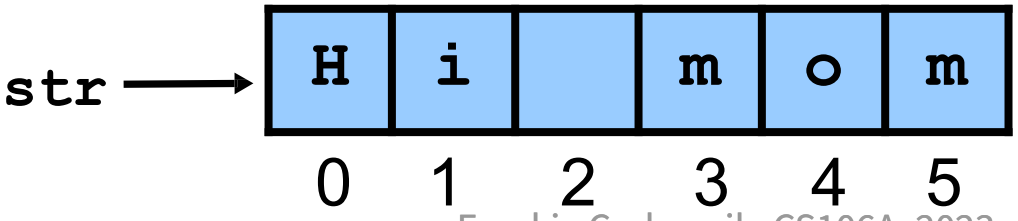


Working with Strings

```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Terminal

```
H
m
6
```

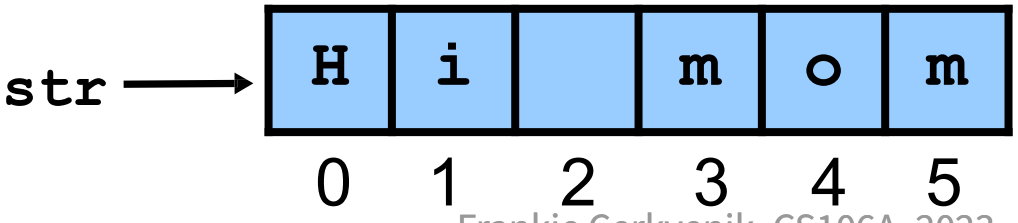


Working with Strings

```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Terminal

```
H
m
6
H
```

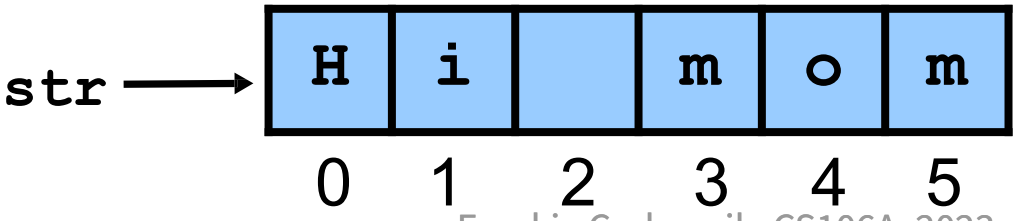


Working with Strings

```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Terminal

```
H
m
6
H
```

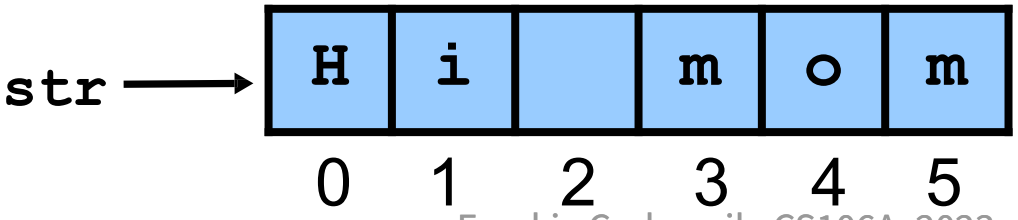


Working with Strings

```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Terminal

```
H
m
6
H
i
```

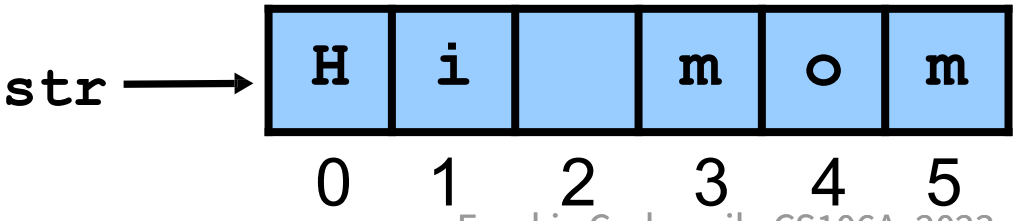


Working with Strings

```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Terminal

```
H
m
6
H
i
```

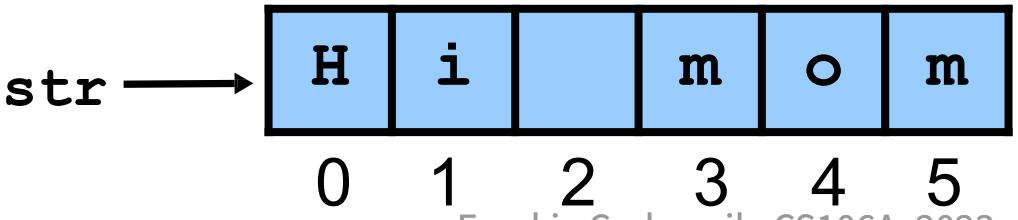


Working with Strings

```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Terminal

```
H
m
6
H
i
```

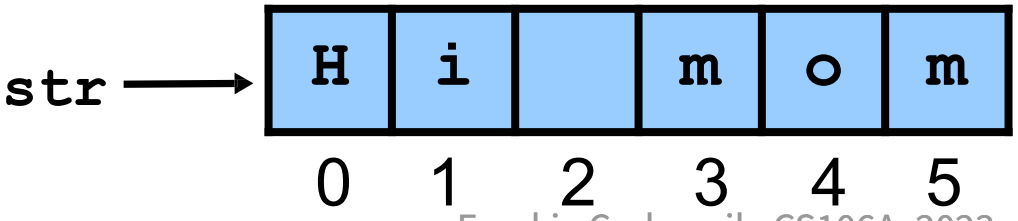


Working with Strings

```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Terminal

```
H
m
6
H
i
```

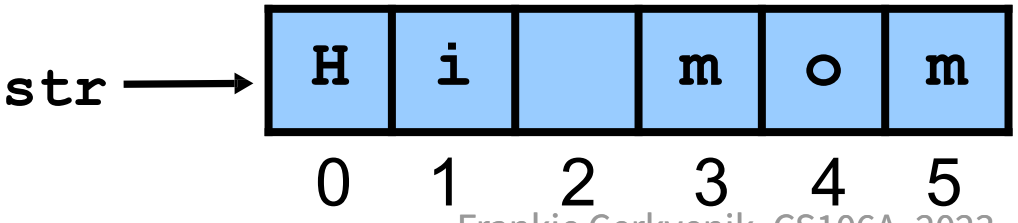


Working with Strings

```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Terminal

```
H
m
6
H
i


m
```

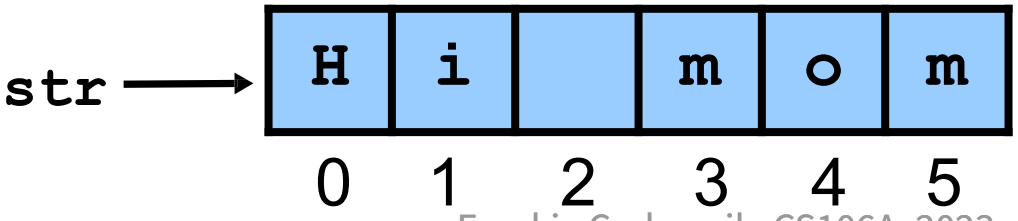


Working with Strings

```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Terminal

```
H
m
6
H
i


m
```

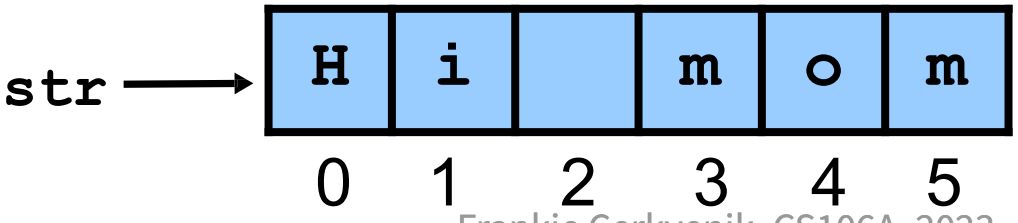


Working with Strings

```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Terminal

```
H
m
6
H
i


m
o
```

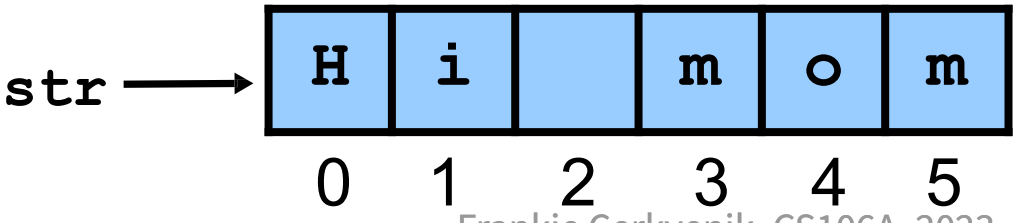


Working with Strings

```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Terminal

```
H
m
6
H
i


m
o
```

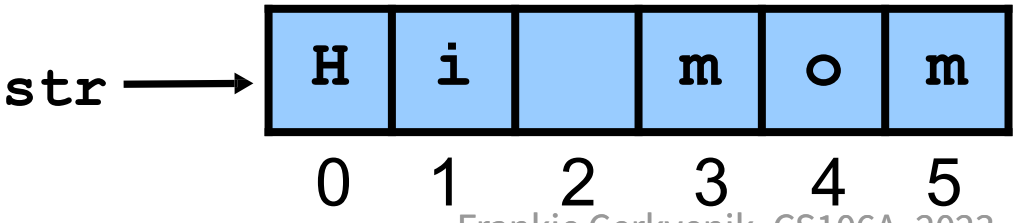


Working with Strings

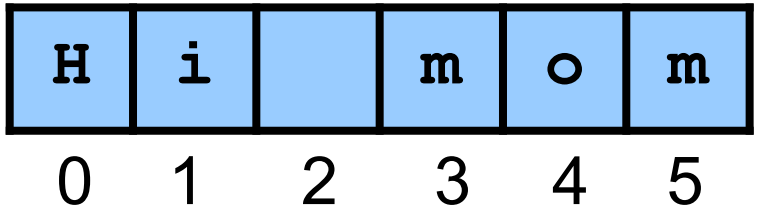
```
def main():
    # create string
    str = "Hi mom"
    # access individual letters
    print(str[0])
    print(str[3])
    # get length
    print(len(str))

    # loop through each letter
    for i in range(len(str)):
        print(str[i])
```

Terminal

```
H
m
6
H
i


m
o
m
```



Key idea!



A string is indexed just like a list!

Slices work too.

It is *almost* like it is a list of characters.

Slicing Strings

- Just like slicing a list, but with a string

  string **[** ***start*** **:** ***end*** **]**

  – Produces a new string with characters from ***string*** starting at index ***start*** up to (but not including) index ***end***

Slicing Strings

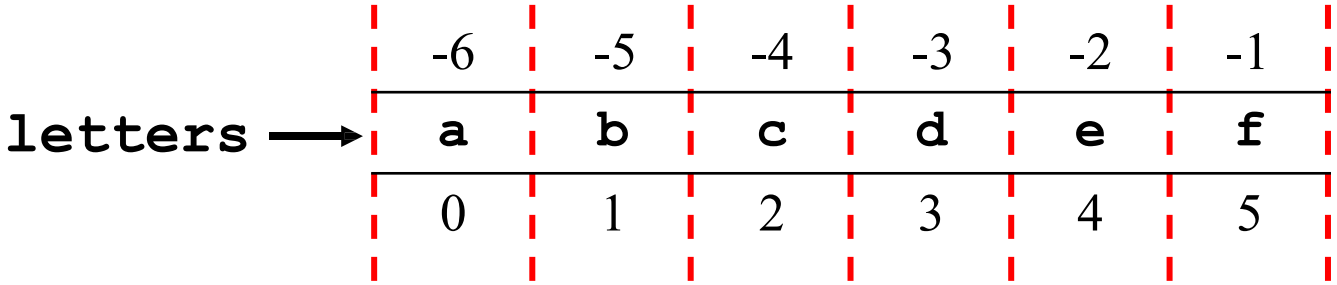
- Just like slicing a list, but with a string

  string **[** ***start*** **:** ***end*** **]**

  – Produces a new string with characters from ***string*** starting at index ***start*** up to (but not including) index ***end***

- Example:

```
letters = 'abcdef'
```




```
letters[2:4]    →   'cd'
letters[1:-2]   →   'bcd'
```

Looping Through Strings

```
name = 'Ecy'
```

- For loop using **`range`**:

```
for i in
    range(len(name)): ch =
    name[i]
    print(ch)
```

Output:

```
E
c
y
```

Looping Through Strings

```
name = 'Ecy'
```

- For loop using **`range`**:

```
for i in
    range(len(name)): ch =
    name[i]
    print(ch)
```

Output:

```
E
c
y
```

- We can also use a "for-each" loop

```
for ch in name:
    print(ch)
```

Output:

```
E
c
y
```

- Both loops iterate over all characters of the string
  - Variable **`ch`** is set to each character in string (in order)

String-a-palooza (String Functions 1)

- Function: ***string1*** + ***string2*** `# concatenation`
  - Returns a new string that is the concatenation of ***string1*** and ***string2***

```
str1 = 'abc'
str2 = 'def'
str3 = str1 + str2 # 'abcdef'
```

String-a-palooza (String Functions 1)

- Function: ***string1*** **+** ***string2*** `# concatenation`
  - Returns a new string that is the concatenation of ***string1*** and ***string2***

```
str1 = 'abc'
str2 = 'def'
str3 = str1 + str2 # 'abcdef'
```

- Function: ***string*****.strip()**
  - Returns a new string with leading/trailing spaces removed

```
message = '   testing  !  '
result = message.strip() #'testing  !'
```

Stanford | ENGINEERING
Computer Science

String-a-palooza (String Functions 2)

- Function: ***string*.split()**
  - Returns a new list based on whitespace separated terms in ***string***

```
str = 'this is a      test!'
list = str.split() # [this, is, a, test!]
```

String-a-palooza (String Functions 2)

- Function: ***string*.split()**
  - Returns a new list based on whitespace separated terms in ***string***

```
str = 'this is a     test!'
list = str.split() # [this, is, a, test!]
```

- Function: ***string*.find(string_to_find)**
  - Returns index of first occurrence of ***string_to_find*** in ***string***
  - Returns -1 if ***string_to_find*** is not found in the original ***string***

```
shout_out = '106A is awesome'
result1 = shout_out.find('is') # result = 5
result2 = shout_out.find('bad')# result = -1
```

String-a-palooza (String Functions 3)

- Function: ***string*.isalpha()**
  - Returns true if all characters in ***string*** are letters in alphabet

```
str1 = 'letters'
result1 = str1.isalpha()# result1 is True
str2 = 'there are spaces here'
result2 = str2.isalpha()# result2 is False
```

String-a-palooza (String Functions 3)

- Function: ***string*.isalpha()**
  - Returns true if all characters in ***string*** are letters in alphabet

```
str1 = 'letters'
result1 = str1.isalpha()# result1 is True
str2 = 'there are spaces here'
result2 = str2.isalpha()# result2 is False
```

- Function: ***string*.isdigit()**
  - Returns true if all characters in ***string*** are digits ('0' to '9')

```
str = '012345'
result = str.isdigit()  # result is True
```

String-a-palooza (String Functions 3)

- Function: ***string*.isalpha()**
  - Returns true if all characters in ***string*** are letters in alphabet

```
str1 = 'letters'
result1 = str1.isalpha()# result1 is True
str2 = 'there are spaces here'
result2 = str2.isalpha()# result2 is False
```

- Function: ***string*.isdigit()**
  - Returns true if all characters in ***string*** are digits ('0' to '9')

```
str = '012345'
result = str.isdigit()  # result is True
```

- Function: ***string*.isspace()**
  - Returns true if all characters in string are whitespace (e.g., space, tab, newline)

```
str = '  '
result = str.isspace()  # result is True
```

String Comparisons

- Can compare strings
  - Test for equal strings (all characters are the same, case sensitive):

```
str = 'abc'
if str == 'abc': # test would be True here
    # body
```

String Comparisons

- Can compare strings
  - Test for equal strings (all characters are the same, case sensitive):

```
str = 'abc'
if str == 'abc': # test would be True here
    # body
```

  - Strings are compared in lexicographic (alphabetic) order:

```
'all' < 'always'   # True
'good' < 'bad'     # False
'table' > 'desk'   # True
```

String Comparisons

- Can compare strings
  - Test for equal strings (all characters are the same, case sensitive):

```
str = 'abc'
if str == 'abc': # test would be True here
    # body
```

  - Strings are compared in lexicographic (alphabetic) order:

```
'all' < 'always'   # True
'good' < 'bad'     # False
'table' > 'desk'   # True
```

  - Case matters (**all** uppercase letters come before lowercase ones)

```
'ABC' < 'abc'      # True
'Zoom' < 'abc'     # True
```

Stanford | ENGINEERING
Computer Science

Strings are Immutable

- Python strings are ***immutable***: once a string has been created **you cannot change its characters**.

```
str = 'kite'
str[0] = 'b' ERROR: Not allowed!!
```

Strings are Immutable

- Python strings are ***immutable***: once a string has been created **you cannot change its characters**.

```
str = 'kite'
str[0] = 'b' ERROR: Not allowed!!
```

- To change a string:
  - ***Create a new string*** holding the new value you want it to have via concatenation.
  - Can *reassign* to the same string variable.

```
str = 'kite'
str = 'b' + str[1:] # str = 'bite'
```

Stanford | ENGINEERING
Computer Science

Strings are Immutable

- Python strings are ***immutable***: once a string has been created **you cannot change its characters**.

- To change a string:
  - ***Create a new string*** holding the new value you want it to have via concatenation.
  - Can *reassign* to the same string variable.

- ***Important consequence:*** if you pass a string into a function, you are guaranteed that string **won't** be changed.
  - Similar to behavior of int and float when passed to a function

Strings are Immutable (Take 1)

```
str = 'abc'
str[1] = 'z'
```

Error!

Traceback (most recent call last):

...

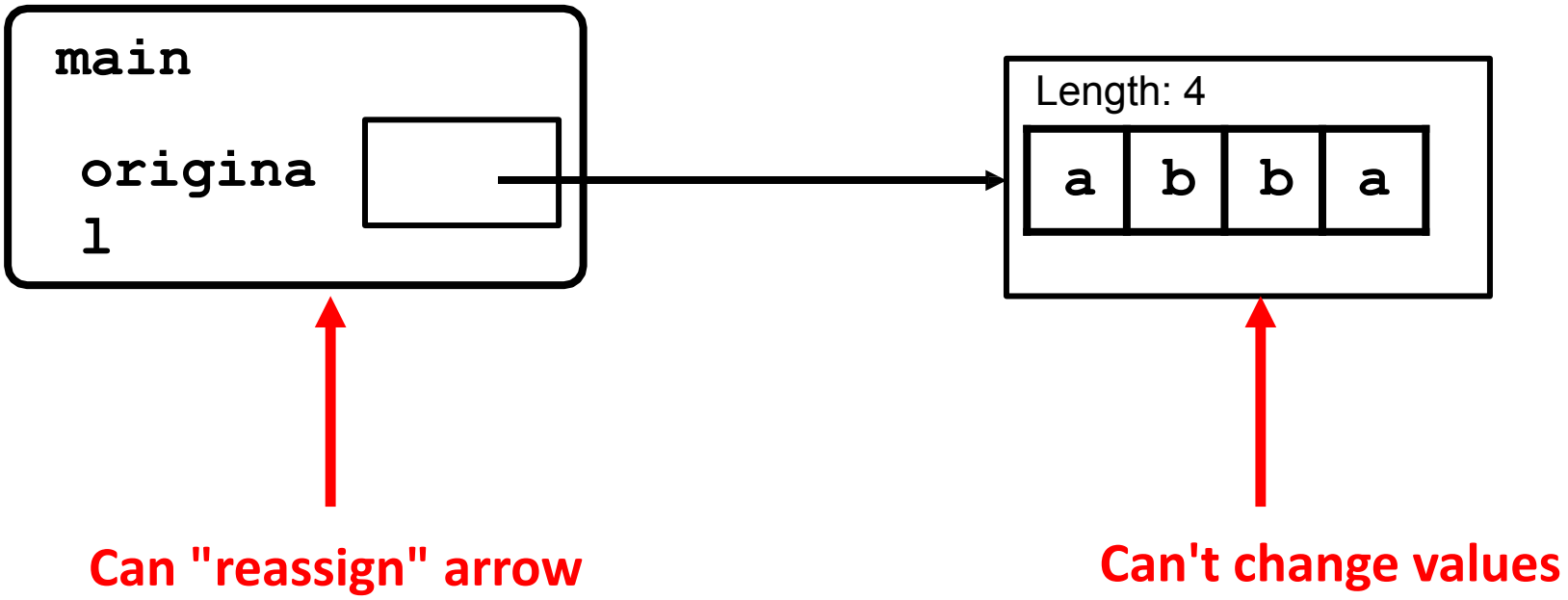
TypeError: 'str' object does not support item assignment

```
str = 'azc'
```

Need to assign a new string

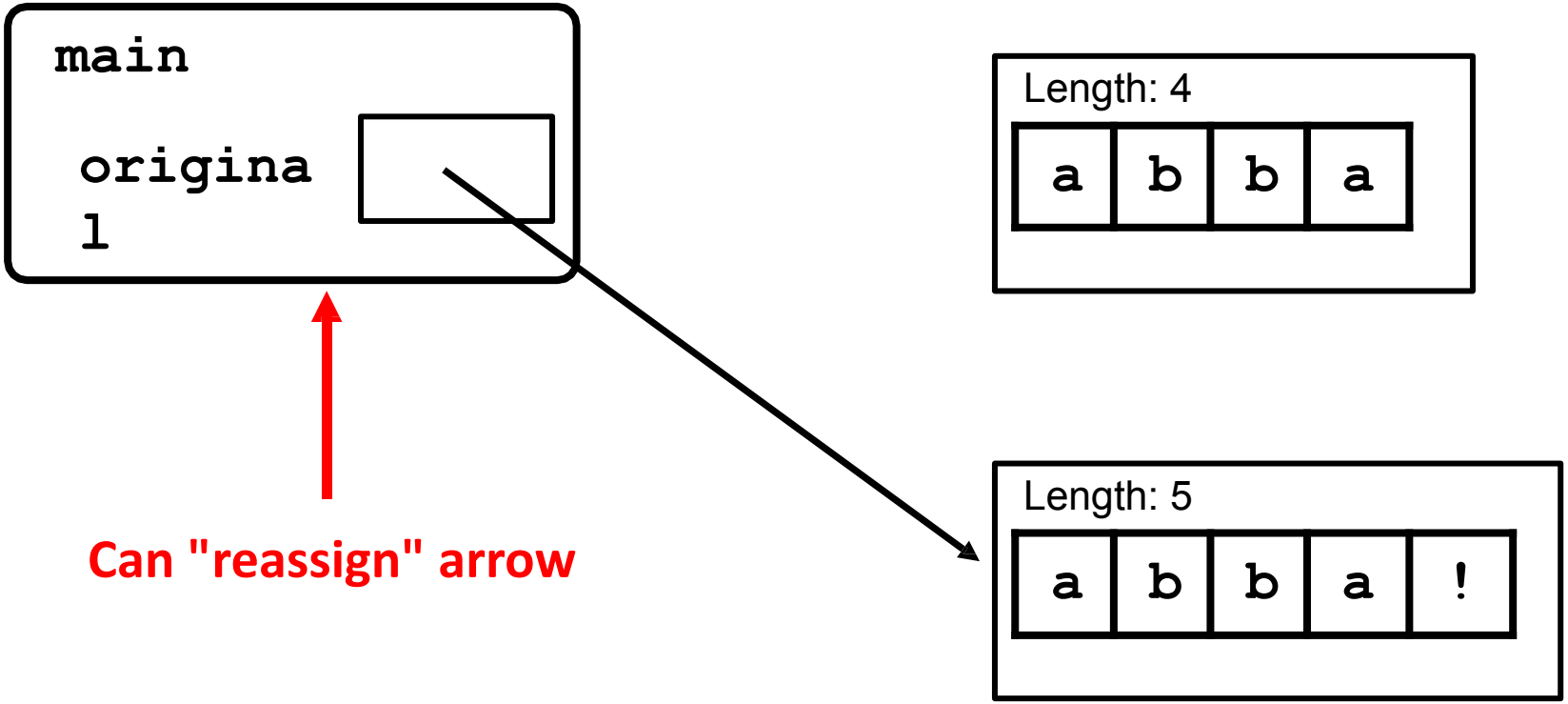
Strings are Immutable (Take 2)

```
original = 'abba'
```



Strings are Immutable (Take 2)

```
original = 'abba'
original += '!'
```



Strings are often made through concatenation

```
def main():
    s1 = "CS106"
    s2 = "A"
    s3 = "I got an " + s2 + " in " + s1 + s2
    print(s3)
```

Terminal:

I got an A in CS106A

The New Hotness: fstrings (formatted strings)

```
year =   2023
action   = 'enjoy CS106A'
important = f'It is {year}, you gotta
{action}!'
print(important)
```

It is 2023, you gotta enjoy CS106A!

Stanford | ENGINEERING
Computer Science

The New Hotness: fstrings (formatted strings)

```
year =   2023
action   = 'enjoy CS106A'
important = f'It is {year}, you gotta
{action}!'
print(important)
```

It is 2023, you gotta enjoy CS106A!

General form:

f'*text* {*variable*} *text* {*variable*} *text* {*variable*}...'

- Values of variables are appended into string
- Can use this anywhere you use strings (print, creating strings, etc.)

Stanford | ENGINEERING
Computer Science

Lists vs Strings

Lists are **mutable**

Strings are **immutable**

Immutable is a guarantee that a function won't be cheeky

(Immutable parameters cannot be modified)

reverse_string

- Write a program that takes in any number of strings as command line arguments and prints each argument reversed

- Start by writing + testing a **`reverse_string`** function, then write main

- **`reverse_string("stressed") -> "desserts"`**

- **`python3 reverse.py 106A rocks`**

  `A601`

  `skcor`

To Pycharm!

reverse.py

Key Idea!

Many string functions use the “loop and construct” pattern

No string functions will be “modify” functions - they will all be “return” functions - be sure to catch those return values!

More string fun

- Write a program that takes in any number of strings as command line arguments
- For each argument, separate out the numerical digits in the string and make a number out of that

  `CS106A CS106B -> 106 106`
- For each argument, separate out the non-numerical digits and make a word/words out of that:

  `CS106A CS106B -> CSA CSB`
- Print the sum of all the numbers and print the words as one long piece of text, separated by spaces

`python3 nums_and_strs.py 106A and 106B`
212
A and B

To Pycharm!

nums_and_strs.py

Recap

- We can process command line arguments as a list

- Text is represented using a string type, which is **like a list of characters whose elements you can't change**

- Strings have lots of built in functions. Reference this slide deck/Google when you need a string function!