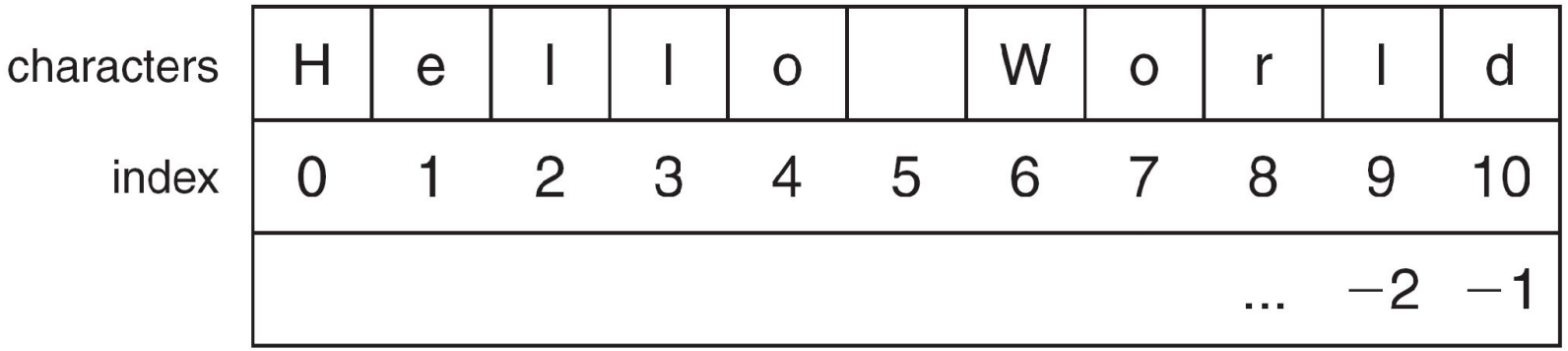
Introduction to Strings

- A string is a sequence of characters.
- A string is denoted by single, double, or triple quotes
- The exact sequence of characters is maintained.

The Index

- Because the elements of a string are a sequence, we can associate each element with an **index**, a location in the sequence:
  - Non-negative values count up from the left, beginning with index 0
  - Negative values count down from the right, starting with -1


characters
H e l l o W o r l d
index
0 1 2 3 4 5 6 7 8 9 10
... −2 −1

Accessing an Element

- A particular element of the string is accessed by the index of the element surrounded by square brackets [ ]

helloStr = ‘Hello World’

print (helloStr[1]) => prints ‘e’

print (helloStr[-1]) => prints ‘d’

print (helloStr[11]) => ERROR

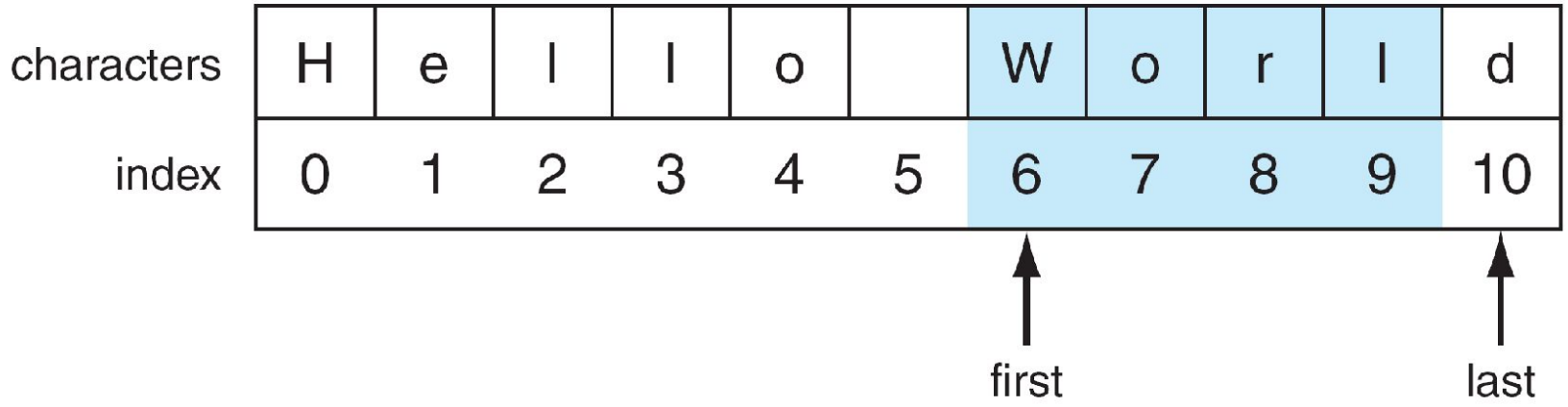
Slicing: the Rules

- Slicing is the ability to select a subsequence of the overall sequence
- Uses the syntax [start : finish], where:
  - start is the index of where we start the subsequence
  - finish is the index of **one after** where we end the subsequence
- If either start or finish are not provided, it defaults to the beginning of the sequence for start and the end of the sequence for finish

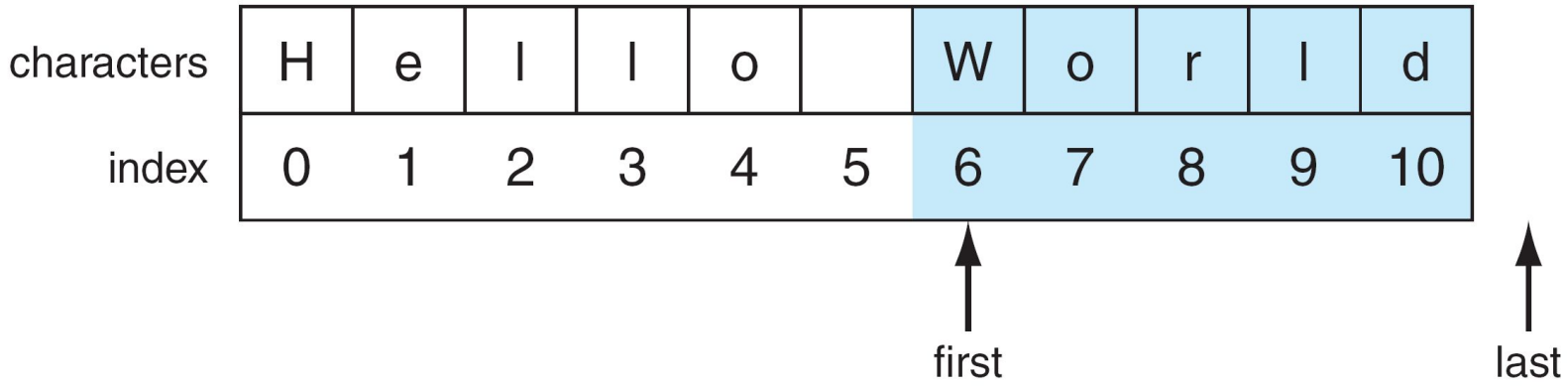
Half Open Range for Slices

- Slicing uses what is called a half-open range
- The first index is included in the sequence
- The last index is one **after** what is included

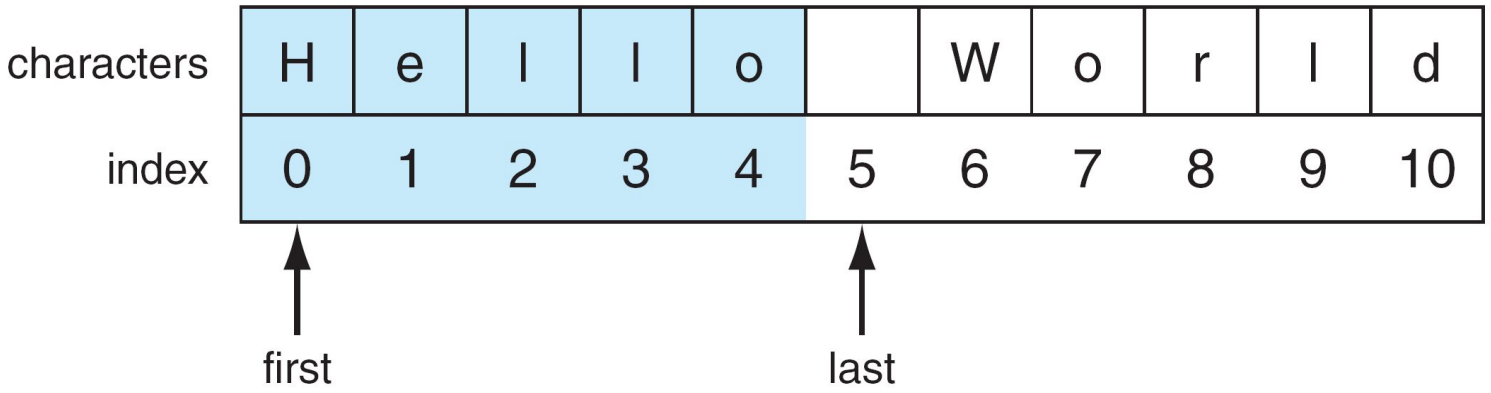
`helloString[6:10]`



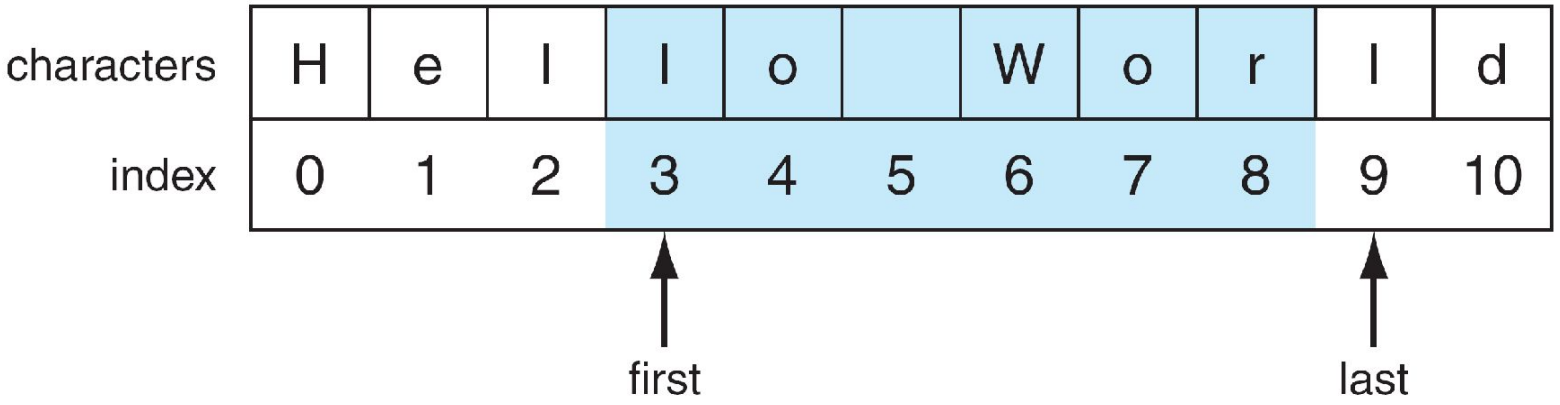
`helloString[6:]`



`helloString[:5]`



```
helloString[3:-2]
```



Basic String Operations

s = ‘spam’

- + is concatenate

newStr = ‘spam’ + ‘-’ + ‘spam-’

print (newStr) ⇒ `spam-spam-`

- * is repeat, the number is how many times

newStr * 3 ⇒

spam-spam-spam-spam-spam-spam-

Some Details

- Both + and * on strings make a new string, but does not modify the arguments.
- Order of operation is important for concatenation and repetition.
- The types required are specific. For concatenation you need two strings; for repetition, a string and an integer.

What Does A + B Mean?

- What operation does the above represent? It depends on the types!
  - two strings, concatenation
  - two integers addition
- The operator + is **overloaded**.
  - the operation + performs depends on the types it is working on

The **type** function

- You can check the type of the value associated with a variable using type

foo = ‘hello world’

type(foo) ⇒ yields <type ‘str’>

foo = 245

type(foo) ⇒ yields <type ‘int’>

Strings are Immutable

- Strings are immutable, that is you cannot change one once you make it:
  - string = ‘spam’
  - string[1] = ‘l’ ⇒ ERROR
- However, you can use it to make another string (copy it, slice it, etc).
  - new_string = string[0] + ‘l’ + string[2:]
  - string ⇒ ‘spam’
  - new_string => ‘slam’

Iteration Through a Sequence

- To date, we have seen the while loop as a way to iterate over a suite (a group of python statements)
- We briefly touched on the for statement for iteration, such as the elements of a list or a string

`for` Statement

We use the for statement to process each element of a list, one element at a time:

```
for item in sequence:
   suite
```

What `for` means

```
string='abc'
for char in string:
    print (char)
```

- first time through, char='a' (string[0])
- second time through, char='b' (string[1])
- third time through, char='c' (string[2])
- no more items in sequence left, we quit

Power of the for Statement

- Sequence iteration as provided by the for statement is very powerful and very useful in Python.
- Allows you to write some very “short” programs that do powerful things.

Built-in function: len

- The len function takes as an argument a string and returns an integer, the length of a string.

```
myStr = 'Hello World'
len(myStr) ⇒  11 # space counts
```

Another version of the `for` loop

```
myStr='abc'
for index in range(len(myStr)):
    print (myStr[index])
```

- first time through, index=0 (myStr[0])
- second time through, index=1 (myStr[1])
- third time through, index=2(myStr[2])
- no more numbers left, so we quit