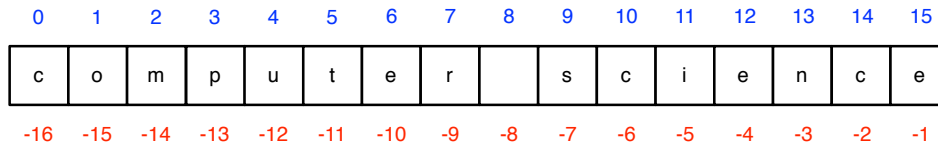


1 String slicing, ord and chr, and methods

A slice is a substring from the start index up to but not including the end index.



```

>>> s = 'computer science'
>>> s[9:16]
'science'
>>> s[9:len(s)]
'science'
>>> s[9:] # default end
'science'
>>> s[:7] # default start
'compute'
>>> s[:] # makes a copy
'computer science'

>>> s[9:12]
'sci'
>>> s[9:-4] # negative slicing
'sci'
>>> s[-7:-4]
'sci'
>>> s[:6] + 'ing' + s[8:]
'computing science'
>>> s[0:len(s):3] # every 3rd char
'cpesee'
>>>
    
```

The **ord** and **chr** functions translate characters to numbers and back. The encoding is based on ASCII (<http://en.wikipedia.org/wiki/ASCII>), an encoding developed at the advent of computing. Example: **ord**('A') returns 65, which happens to be the ASCII encoding of 'A'; **chr**(65) returns 'A'.

String Method	Arguments	Description
lower	none	Returns a lowercase copy
upper	none	Returns an uppercase copy
capitalize	none	Returns a copy with first letter capitalized
isalpha	none	Returns True if all characters are alphabetic
isdigit	none	Returns True if all characters are digits
endswith	sub	Returns True if string ends with sub
startswith	sub	Returns True if string starts with sub
find	sub, start, end	Returns the lowest index where substring sub occurs or -1 if no occurrences. The start and end parameters are <i>optional</i> .
count	sub, start, end	Returns the number of non-overlapping occurrences of the substring sub in the string. The start and end parameters are <i>optional</i> .
replace	old, new	Returns a copy of string with every occurrence of substring old replaced with substring new.
strip	none	Returns a copy with leading and trailing whitespace removed
split	sep	Returns a list of “words” in string, splitting on whitespace. If <i>optional</i> separator sep is given, it splits on that instead.
join	list of strings	Returns a string that consists of the strings in list joined together with this string separating them.

A table of string *methods* followed by examples. For a complete list of methods, type `dir(str)` into the IDLE shell and then use `help` to get more information, as in `help(str.join)`. Remember: strings are immutable so none of these methods *change* the string: they return *copies*.

```
>>> s = 'banana!'
>>> s.find('!')
6
>>> s.find('!', 3, 6) # up to but not including end
-1
>>> 'banana!'.count('a')
3
>>> s.count('ana') # non-overlapping
1
>>> s.replace('an', 'wah')
'bwahwaha!'
>>> s = '    spacey    banana    '
>>> s.strip()
'spacey    banana'
>>> s = 'hello        jello        fellow'
>>> s.split()
['hello', 'jello', 'fellow']
>>> some_data = "Joe Student;12/17/1995;Ithaca,NY;3.6"
>>> some_data.split(';')
['Joe Student', '12/17/1995', 'Ithaca,NY', '3.6']
>>> ', '.join(['apple', 'lemon', 'pear'])
'apple, lemon, pear'
```

2 Exercises

Solutions are presented in class and also included in the moodle version of this handout.

1. Assume variable `s` is a string that refers to someone's full name, as in `'Tom Brady'`. Write a **single expression** that produces the name with the first name abbreviated, as in `'T. Brady'`. Hint: Use the `find` and slicing.

Solution:

```
s = 'Tom Brady'
print s[0] + '.' + s[s.find(' ')+1:]

s = 'Peyton Manning'
print s[0] + '.' + s[s.find(' ')+1:]
```

2. Assume variables `s1` and `s2` refer to strings. Write a **single expression** that produces the index of the *second* occurrence of `s2` in `s1`. If `s2` does not occur twice in `s1`, the expression should

produce -1. For example, if `s1` is "banana" and `s2` is "ana", your expression should return 3. You cannot use indexing nor slicing; you can use `find` and arithmetic. Hint: call `find` twice.

Solution:

```
s1 = 'banana'
s2 = 'ana'
print s1.find(s2, s1.find(s2) + 1)

s1 = 'apple'
s2 = 'p'
print s1.find(s2, s1.find(s2) + 1)
```

3. Write a function `acronym` that takes a string representing a phrase or name and returns the equivalent acronym, upper-cased. Example: if the input is 'computer science investigation', it should return 'C.S.I.'. Hint: use a `for` loop and the `split` and `join` and `upper` methods.

Solution:

```
def acronym(s):
    '''(str) -> str
    Expects s to be a string of several words.
    Returns the corresponding acronym consisting
    of initial of each word in uppercase, separated
    by periods.
    >>> acronym('computer science investigation')
    'C.S.I.'
    >>> acronym('abbreviated coded rendition of name yielding meaning')
    'A.C.R.O.N.Y.M.'
    '''
    words = s.split()
    initials = []
    for word in words:
        initials += [word[0]]
    return '.'.join(initials).upper() + '.'
```

4. Write a function `mycount` that takes two strings `s` and `sub` and returns the number of *nonoverlapping* occurrences of `sub` in `s`. You cannot use the `count` method. Hint: use a `while` loop and the `find` method, making use of the optional `start` parameter of `find`.

Solution:

version 1: uses a loop and a half

```
def mycount(s, sub):
    '''(str, str) -> int
    Returns the number of non-overlapping occurrences
    of sub in s.
    >>> mycount('apple', 'p')
    2
    >>> mycount('apple', 'P')
    0
    >>> mycount('banana', 'ana')
    1
    >>> mycount('aaa', 'aa')
    1
    >>> mycount('late for a date with a mate', 'ate')
    3
    '''
    count = 0
    idx = s.find(sub) # half a loop
    while idx > -1:
        count += 1
        idx = s.find(sub, idx+len(sub))
    return count
```

version 2: uses an infinite loop with break

```
def mycount2(s, sub):
    '''(str, str) -> int
    Returns the number of non-overlapping occurrences
    of sub in s.
    >>> mycount2('apple', 'p')
    2
    >>> mycount2('apple', 'P')
    0
    >>> mycount2('banana', 'ana')
    1
    >>> mycount2('aaa', 'aa')
    1
    >>> mycount2('late for a date with a mate', 'ate')
    3
    '''
    count = 0
    idx = 0
    while True:
```

```
    idx = s.find(sub, idx)
    if idx == -1:
        break
    count += 1
    idx += len(sub)
return count
```

Some material adapted from Campbell and Gries.