Lecture 3. Python Primer – Part 3

Chao Chen

Stony Brook University

Sept. 26, 2022

Variable initialization and updating

- Variable update (increment/decrement)

```
x = x + 1
```

- If have not seen x before

```
>>> x = x + 1
NameError: name 'x' is not defined
```

- If have not seen x before, initialize with a simple assignment

```
>>> x = 0
>>> x = x + 1
```

Need for automatically repeated updating

- Initialize n first, decrement n at each iteration, until n == 0

```
n = 5
while n > 0:
    print(n)
    n = n - 1
print('Blastoff!')
```

1. Evaluate the condition, yielding `True` or `False`.
2. If the condition is false, exit the `while` statement and continue execution at the next statement.
3. If the condition is true, execute the body and then go back to step 1.

Infinite loops

1. Evaluate the condition, yielding `True` or `False`.
2. If the condition is false, exit the `while` statement and continue execution at the next statement.
3. If the condition is true, execute the body and then go back to step 1.

```
n = 5
while n > 0:
    print(n)
    n = n - 1
print('Blastoff!')
```

Infinite loops (cont'd)

- What will happen?

```
n = 10
while True:
    print(n, end=' ')
    n = n - 1
print('Done!')
```

- Use break to stop the loop!

```
while True:
    line = input('> ')
    if line == 'done':
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> finished
finished
> done
Done!
```

Infinite loops (cont'd)

- Use both break and continue to control the flow
- Break: stop the current loop
- Continue: stop the current iteration, continue with the next iteration

```python
while True:
    line = input('> ')
    if line[0] == '#':
        continue
    if line == 'done':
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> # don't print this
> print this!
print this!
> done
Done!
```

definite loops

```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends:
    print('Happy New Year:', friend)
print('Done!')
```

```
Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
Done!
```

`friend` is the *iteration variable* :

- Initializing one or more variables before the loop starts
- Performing some computation on each item in the loop body, possibly changing the variables in the body of the loop
- Looking at the resulting variables when the loop completes

definite loops (cont'd)

- What does this code do?

```
count = 0
for itervar in [3, 41, 12, 9, 74, 15]:
    count = count + 1
print('Count: ', count)
```

- What does this code do?

```
total = 0
for itervar in [3, 41, 12, 9, 74, 15]:
    total = total + itervar
print('Total: ', total)
```

definite loops (cont'd)

- Example, finding the largest value

```
largest = None
print('Before:', largest)
for itervar in [3, 41, 12, 9, 74, 15]:
    if largest is None or itervar > largest :
        largest = itervar
    print('Loop:', itervar, largest)
print('Largest:', largest)
```

```
Before: None
Loop: 3 3
Loop: 41 41
Loop: 12 41
Loop: 9 41
Loop: 74 74
Loop: 15 74
Largest: 74
```

- What if forgot to initialize largest, and it was used before? (demo)
- Break / continue still work

definite loops (cont’d)

- Example, finding the smallest value

```
smallest = None
print('Before:', smallest)
for itervar in [3, 41, 12, 9, 74, 15]:
    if smallest is None or itervar < smallest:
        smallest = itervar
    print('Loop:', itervar, smallest)
print('Smallest:', smallest)
```

definite loops (cont'd)

- Example, finding the smallest value – as a function

```
def min(values):
    smallest = None
    for value in values:
        if smallest is None or value < smallest:
            smallest = value
    return smallest
```

Exercise

Exercise 1: Write a program which repeatedly reads numbers until the user enters "done". Once "done" is entered, print out the total, count, and average of the numbers. If the user enters anything other than a number, detect their mistake using `try` and `except` and print an error message and skip to the next number.

```
Enter a number: 4
Enter a number: 5
Enter a number: bad data
Invalid input
Enter a number: 7
Enter a number: done
16 3 5.333333333333333
```

Exercise

Exercise 2: Write another program that prompts for a list of numbers as above and at the end prints out both the maximum and minimum of the numbers instead of the average.

String

A string is a *sequence* of characters. You can access the characters one at a time with the bracket operator:

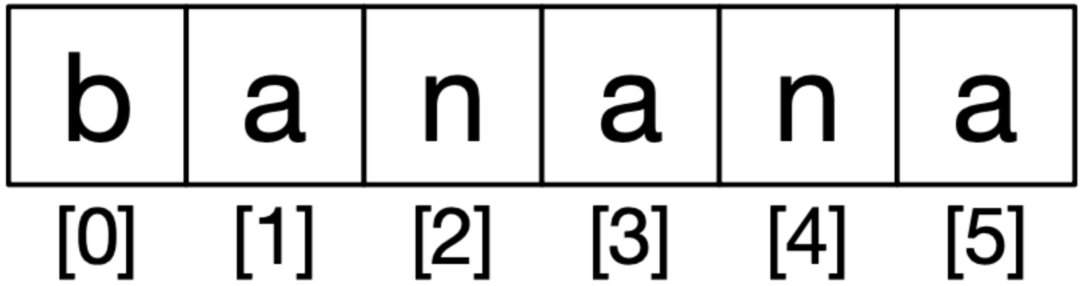
```
>>> fruit = 'banana'
>>> letter = fruit[1]
```

```
>>> print(letter)
a
```

```
>>> letter = fruit[0]
>>> print(letter)
b
```

String

So b is the 0th letter ("zero-eth") of "banana", a is the 1th letter ("one-eth"
n is the 2th ("two-eth") letter.



```
>>> letter = fruit[1.5]
TypeError: string indices must be integers
```

String

`len` is a built-in function that returns the number of characters in a string:

```
>>> fruit = 'banana'
>>> len(fruit)
6
```

```
>>> length = len(fruit)
>>> last = fruit[length]
IndexError: string index out of range
```

```
>>> last = fruit[length-1]
>>> print(last)
a
```

- Or you can use
  - fruit[-1] (the last character),
  - fruit[-2] (second from the last)

String

- Loop through a string

```
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(letter)
    index = index + 1
```

Exercise 1: Write a `while` loop that starts at the last character in the string and works its way backwards to the first character in the string, printing each letter on a separate line, except backwards.

String

Another way to write a traversal is with a **for** loop:

```
for char in fruit:
    print(char)
```

Each time through the loop, the next character in the string is assigned to the variable **char**. The loop continues until no characters are left.

String - slice

A segment of a string is called a *slice*. Selecting a slice is similar to selecting a character:

```
>>> s = 'Monty Python'
>>> print(s[0:5])
Monty
>>> print(s[6:12])
Python
```

String – slice

If you omit the first index (before the colon), the slice starts at the beginning of the string. If you omit the second index, the slice goes to the end of the string:

```
>>> fruit = 'banana'
>>> fruit[:3]
'ban'
>>> fruit[3:]
'ana'
```

The operator returns the part of the string from the “n-eth” character to the “m-eth” character, including the first but excluding the last.

String

If the first index is greater than or equal to the second the result is an *empty string*, represented by two quotation marks:

```
>>> fruit = 'banana'
>>> fruit[3:3]
''
```

An empty string contains no characters and has length 0, but other than that, it is the same as any other string.

Exercise 2: Given that **`fruit`** is a string, what does **`fruit[:]`** mean?

String

- String is immutable – cannot change its content

```
>>> greeting = 'Hello, world!'
>>> greeting[0] = 'J'
TypeError: 'str' object does not support item assignment
```

- Example 2

```
>>> greeting = 'Hello, world!'
>>> new_greeting = 'J' + greeting[1:]
>>> print(new_greeting)
Jello, world!
```

- The assignment creates a new string using ‘J’ and a slice of greeting

Counting a letter in a string

```
word = 'banana'
count = 0
for letter in word:
    if letter == 'a':
        count = count + 1
print(count)
```

Exercise 3:

Encapsulate this code in a function named **`count`**, and generalize it so that it accepts the string and the letter as arguments.

Checking if a Substring Exists

```
>>> 'a' in 'banana'
True
>>> 'seed' in 'banana'
False
```

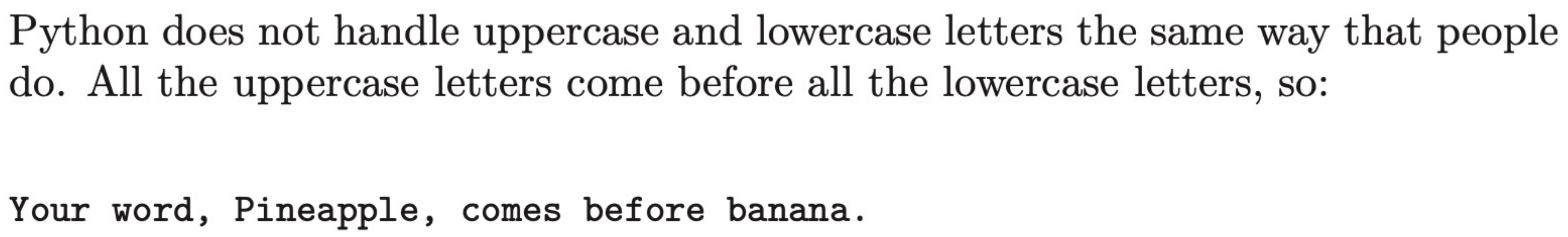
Comparing strings

```
if word == 'banana':
    print('All right, bananas.')
```

- Comparison in terms of alphabetic order

```
if word < 'banana':
    print('Your word,' + word + ', comes before banana.')
elif word > 'banana':
    print('Your word,' + word + ', comes after banana.')
else:
    print('All right, bananas.')
```

Comparison



Python does not handle uppercase and lowercase letters the same way that people do. All the uppercase letters come before all the lowercase letters, so:

```
Your word, Pineapple, comes before banana.
```

- To resolve the issue: first lowercase the whole word, then compare how?

Strings are objects

Strings are an example of Python *objects*. An object contains both data (the actual string itself) and *methods*, which are effectively functions that are built into the object and are available to any *instance* of the object.

Python has a function called `dir` which lists the methods available for an object. The `type` function shows the type of an object and the `dir` function shows the available methods.

```
>>> stuff = 'Hello world'
>>> type(stuff)
<class 'str'>
>>> dir(stuff)
['capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'maketrans', 'partition', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
'split', 'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
>>> help(str.capitalize)
Help on method_descriptor:

capitalize(...)
    S.capitalize() -> str

    Return a capitalized version of S, i.e. make the first character
    have upper case and the rest lower case.
>>>
```

Calling a method

- Calling a *method* is similar to calling a function (it takes arguments and returns a value) but the syntax is different. We call a method by appending the method name to the variable name using the period as a delimiter.

```
>>> word = 'banana'
>>> new_word = word.upper()
>>> print(new_word)
BANANA
```

Find

```
>>> word = 'banana'
>>> index = word.find('a')
>>> print(index)
1
```

```
>>> word.find('na')
2
```

- What if I want to find the second match?

- The second argument specify starting location/index

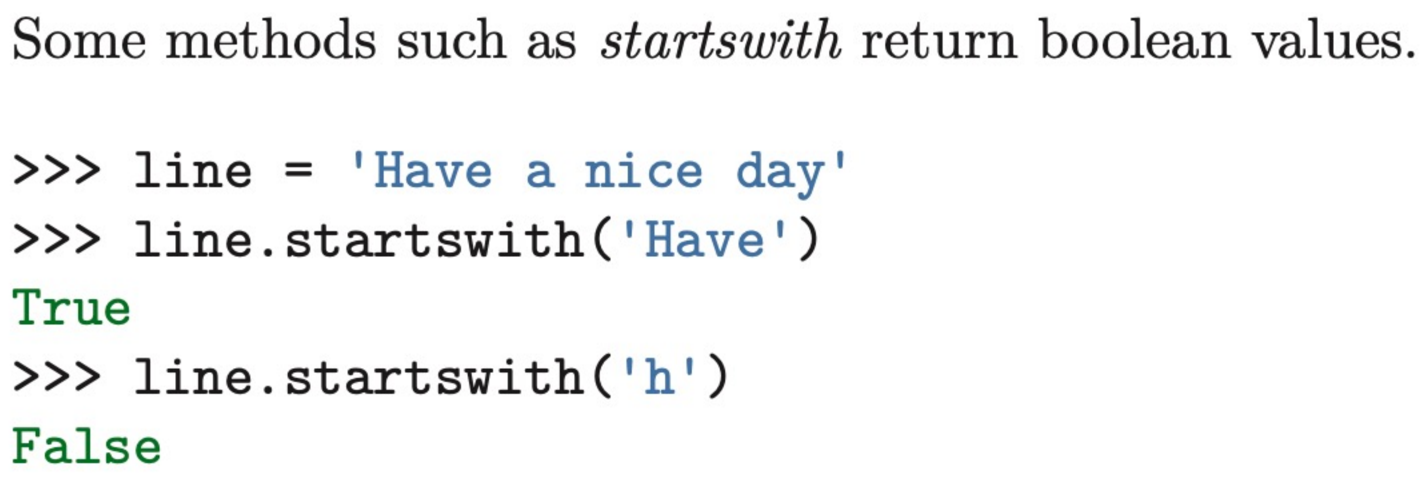
```
>>> word.find('na', 3)
4
```

Removing white space

One common task is to remove white space (spaces, tabs, or newlines) from the beginning and end of a string using the **`strip`** method:

```
>>> line = '  Here we go  '
>>> line.strip()
'Here we go'
```

Startswith

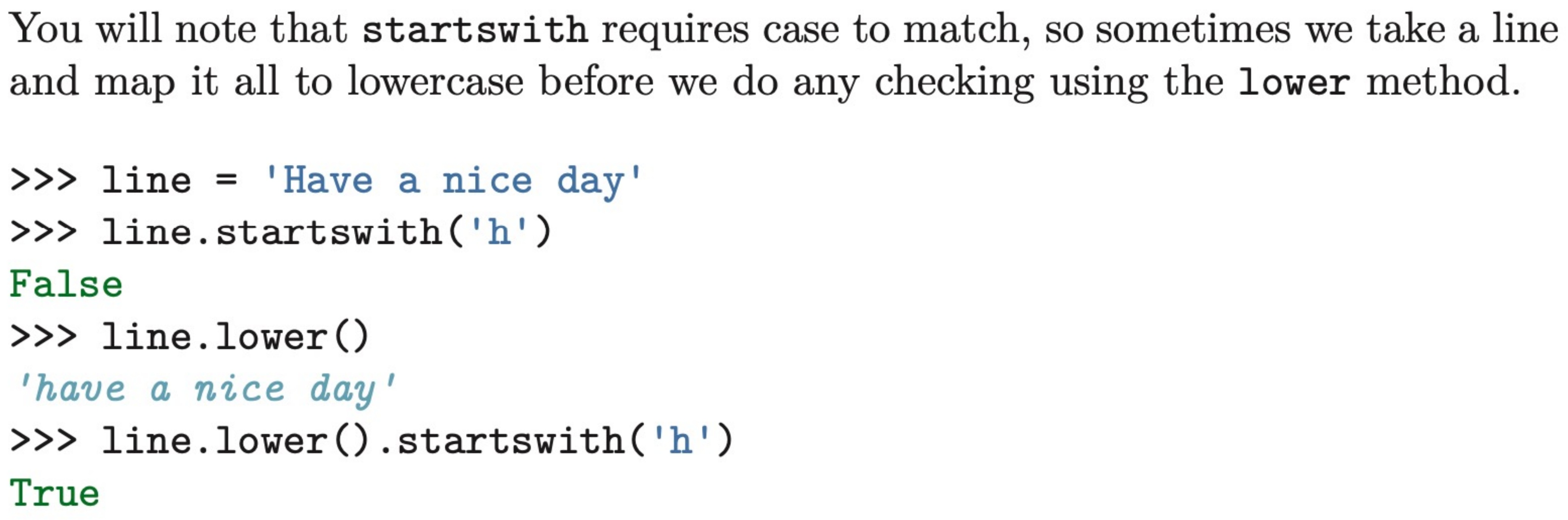


Some methods such as *startswith* return boolean values.

```
>>> line = 'Have a nice day'
>>> line.startswith('Have')
True
>>> line.startswith('h')
False
```

- What if I do not want to be case sensitive?

Startswith



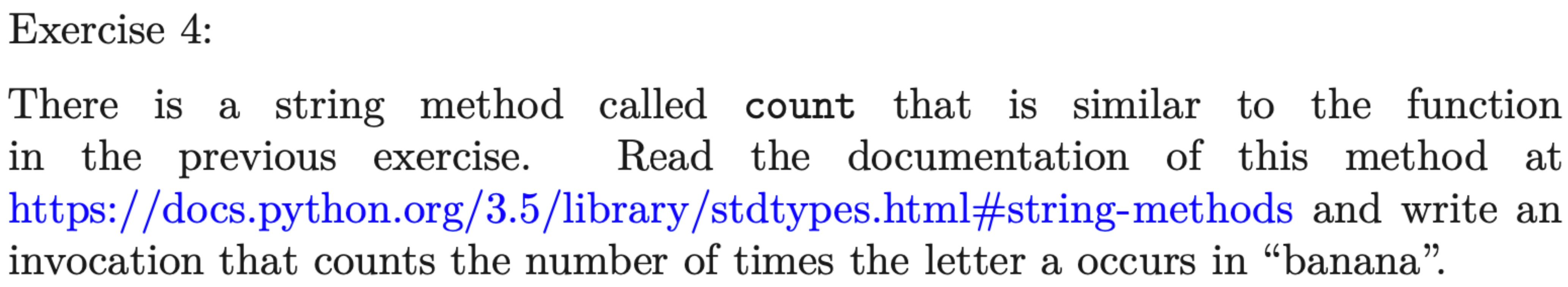
- You will note that **`startswith`** requires case to match, so sometimes we take a line and map it all to lowercase before we do any checking using the **`lower`** method.

```
>>> line = 'Have a nice day'
>>> line.startswith('h')
False
>>> line.lower()
'have a nice day'
>>> line.lower().startswith('h')
True
```

Check out functions

- Learn to use library references to find all methods

  https://docs.python.org/3/library/stdtypes.html#string-methods



Exercise 4:

There is a string method called **`count`** that is similar to the function in the previous exercise. Read the documentation of this method at https://docs.python.org/3.5/library/stdtypes.html#string-methods and write an invocation that counts the number of times the letter a occurs in "banana".

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

and we wanted to pull out only the second half of the address (i.e., `uct.ac.za`) from each line, we can do this by using the `find` method and string slicing.

First, we will find the position of the at-sign in the string. Then we will find the position of the first space *after* the at-sign. And then we will use string slicing to extract the portion of the string which we are looking for.

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print(atpos)
21
>>> sppos = data.find(' ',atpos)
>>> print(sppos)
31
>>> host = data[atpos+1:sppos]
>>> print(host)
uct.ac.za
>>>
```

Formatting

- Format operator ‘%’ – replace part of a string with the data in another variable (can be any type)
- Note this is different from modulus operator in arithmetic operations\
- %d – substitute in an integer

```
>>> camels = 42
>>> '%d' % camels
'42'
```

```
>>> camels = 42
>>> 'I have spotted %d camels.' % camels
'I have spotted 42 camels.'
```

- Once substituted in, ‘42’ is part of the string, not an integer any more

Formatting

- %g – float, %s – string

```
>>> 'In %d years I have spotted %g %s.' % (3, 0.1, 'camels')
'In 3 years I have spotted 0.1 camels.'
```

- Numbers and types must match

```
>>> '%d %d %d' % (1, 2)
TypeError: not enough arguments for format string
>>> '%d' % 'dollars'
TypeError: %d format: a number is required, not str
```

Debugging

- To produce robust program, always think like an adversary: what could one do to crack the code
- How to crack the following code

```
while True:
    line = input('> ')

    if line[0] == '#':
        continue
    if line == 'done':
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> # don't print this
> print this!
print this!
>
Traceback (most recent call last
  File "copytildone.py", line 3,
    if line[0] == '#':
IndexError: string index out of 
```

```
# Code: http://www.py4e.com/code3/copytildone2.py
```

Alternative solution

One possibility is to simply use the **startswith** method which returns **False** if the string is empty.

```
    if line.startswith('#'):
```

Another way is to safely write the **if** statement using the *guardian* pattern and make sure the second logical expression is evaluated only where there is at least one character in the string.:

```
    if len(line) > 0 and line[0] == '#':
```

Exercises

Exercise 5: Take the following Python code that stores a string:‘

```
str = ’X-DSPAM-Confidence:0.8475’
```

Use **`find`** and string slicing to extract the portion of the string after the colon character and then use the **`float`** function to convert the extracted string into a floating point number.

Exercises

Exercise 6:

Read the documentation of the string methods at

https://docs.python.org/3/library/stdtypes.html#string-methods

You might want to experiment with some of them to make sure you understand how they work. `strip` and `replace` are particularly useful.

The documentation uses a syntax that might be confusing. For example, in `find(sub[, start[, end]])`, the brackets indicate optional arguments. So `sub` is required, but `start` is optional, and if you include `start`, then `end` is optional.

Check out split, replace, join function, it is very useful

The end