Quiz #1

- Rewrite the grade program from the previous slides using a function called **computegrade** that takes a score as its parameter and returns a grade as a string.
  - Score   Grade
  - > 0.9   A
  - > 0.8   B
  - > 0.7   C
  - > 0.6   D
  - <= 0.6   F

Iteration

Updating variables

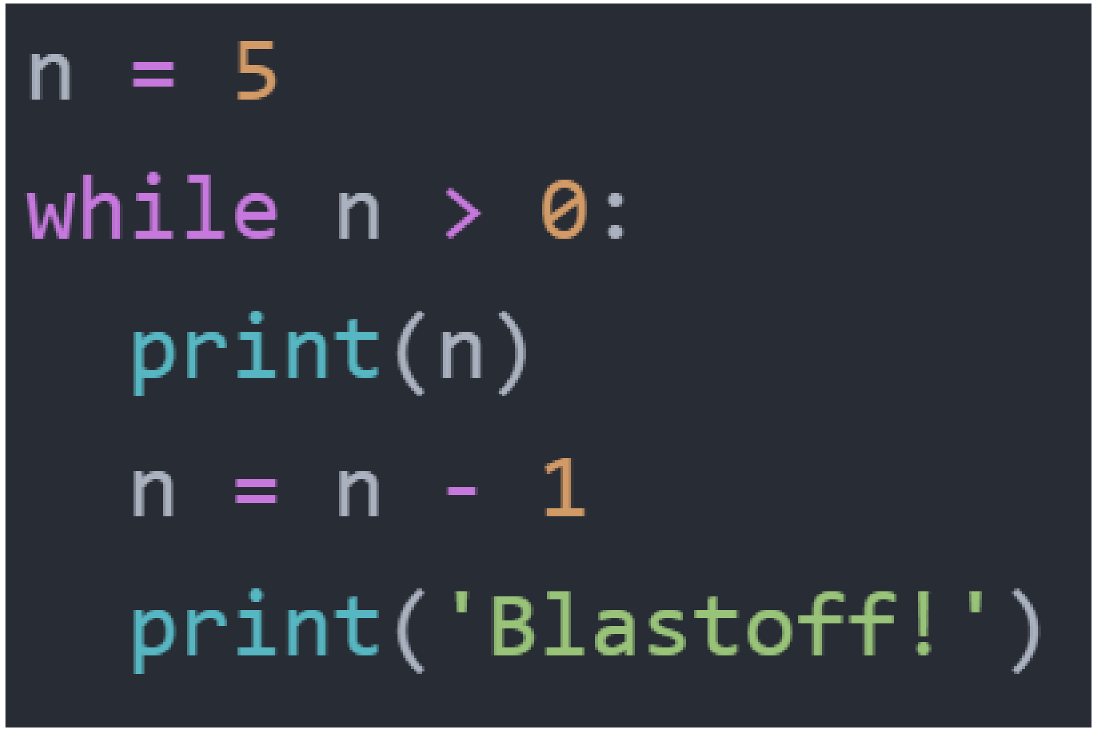
- A common pattern in assignment statements is an assignment statement that updates a variable, where the new value of the variable depends on the old

```
x = x + 1
```

The while statement

- Computers are often used to automate repetitive tasks
- Repeating identical or similar tasks without making errors is something that computers do well and people do poorly
- One form of iteration in Python is the **while** statement

The while statement



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

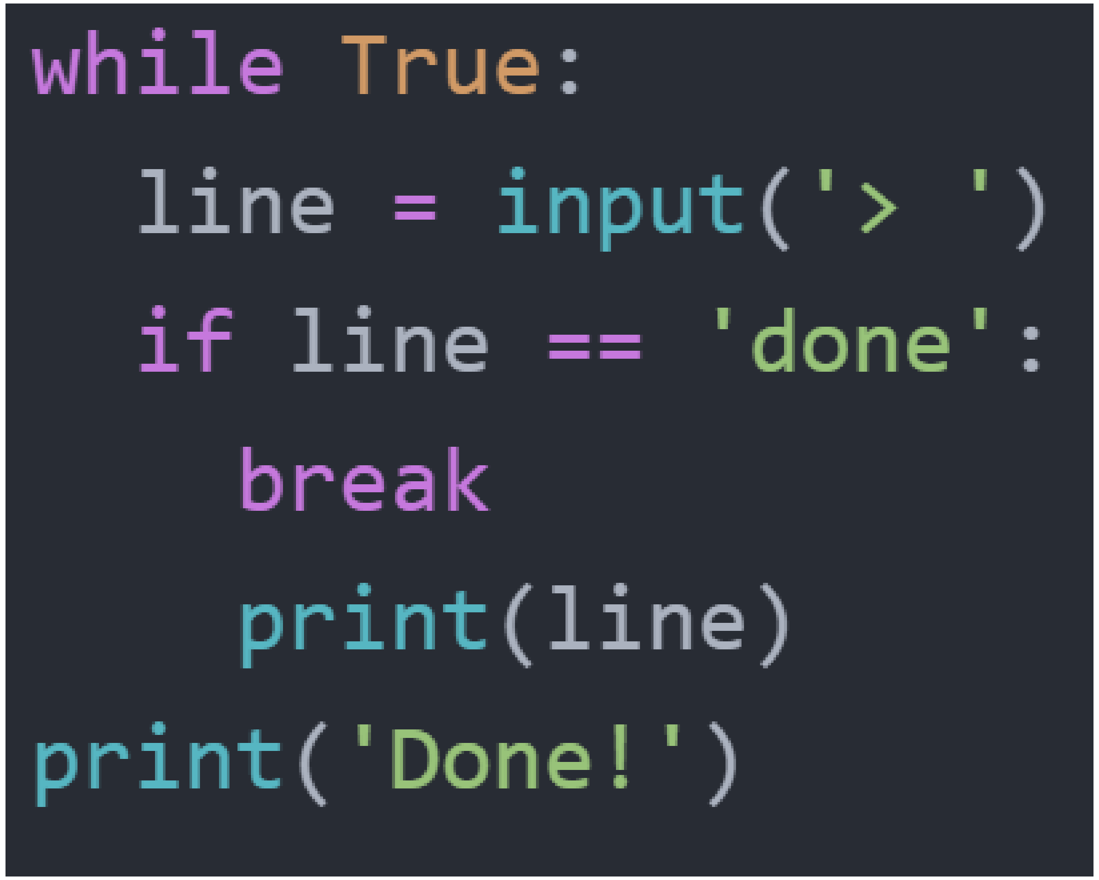
The while statement

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

Infinite loops

- Sometimes you don’t know it’s time to end a loop until you get half way through the body
- In that case you can write an infinite loop on purpose and then use the **break** statement to jump out of the loop

Break Statement



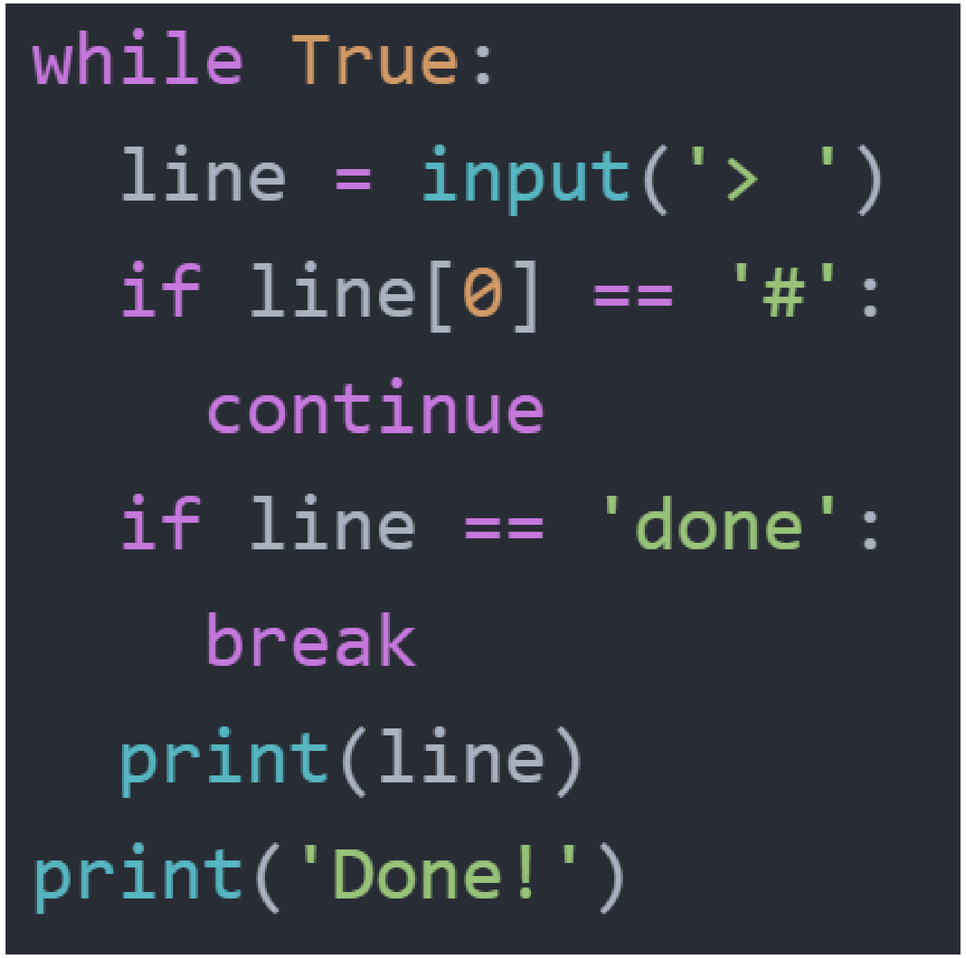
```python
while True:
  line = input('> ')
  if line == 'done':
    break
    print(line)
print('Done!')
```

Finishing iterations with continue

- Sometimes you are in an iteration of a loop and want to finish the current iteration and immediately jump to the next iteration

- In that case you can use the **continue** statement to skip to the next iteration without finishing the body of the loop for the current iteration

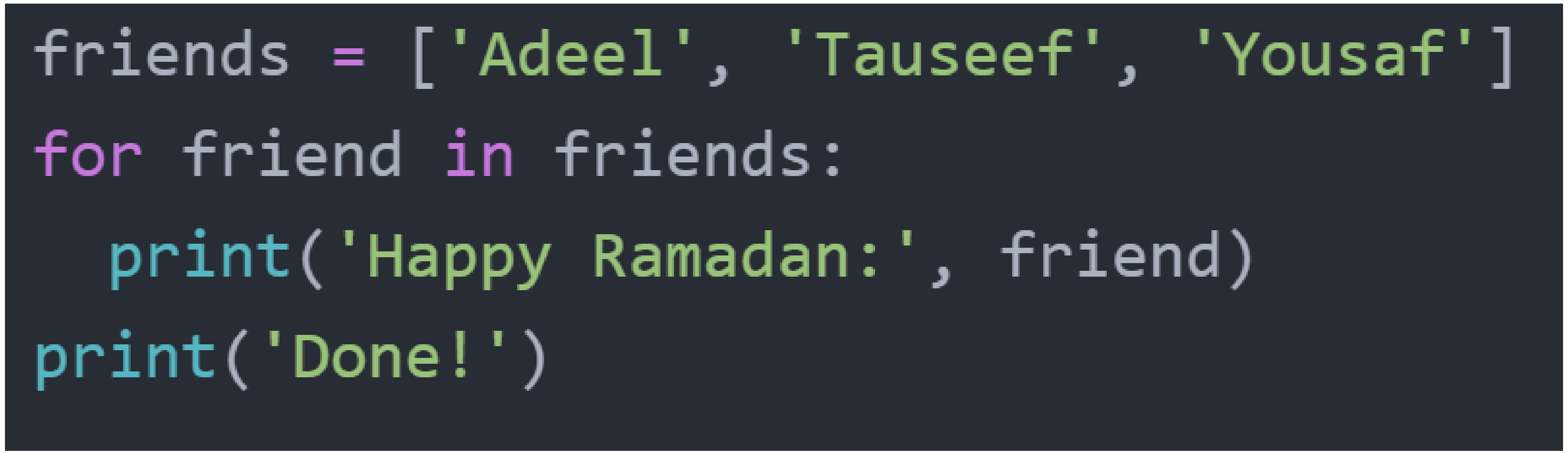
Continue Statement


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

Definite loops using for

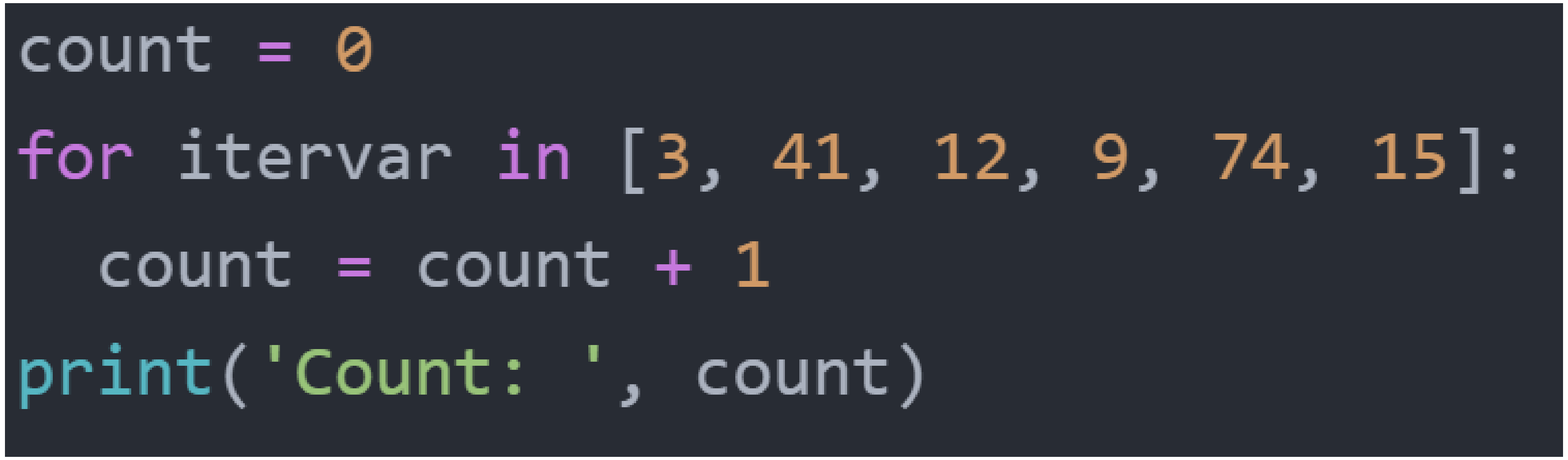
- Sometimes we want to loop through a set of things such as a list of words, the lines in a file, or a list of numbers
- When we have a list of things to loop through, we can construct a definite loop using a for statement
- We call the while statement an indefinite loop because it simply loops until some condition becomes False, whereas the for loop is looping through a known set of items so it runs through as many iterations as there are items in the set

for Loop



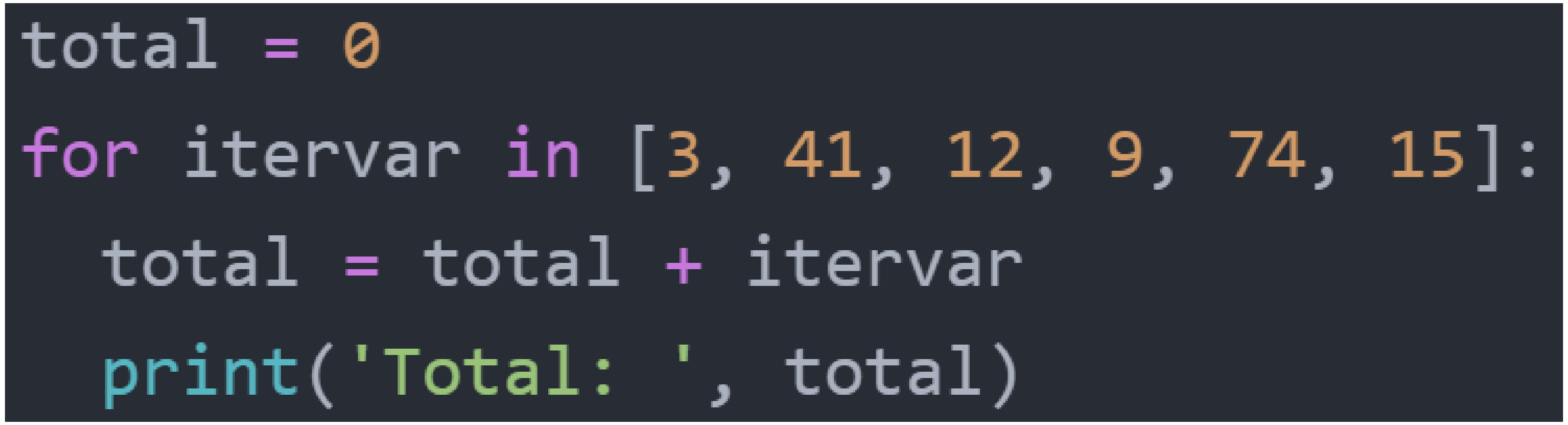
```python
friends = ['Adeel', 'Tauseef', 'Yousaf']
for friend in friends:
  print('Happy Ramadan:', friend)
print('Done!')
```

Counting and summing loops



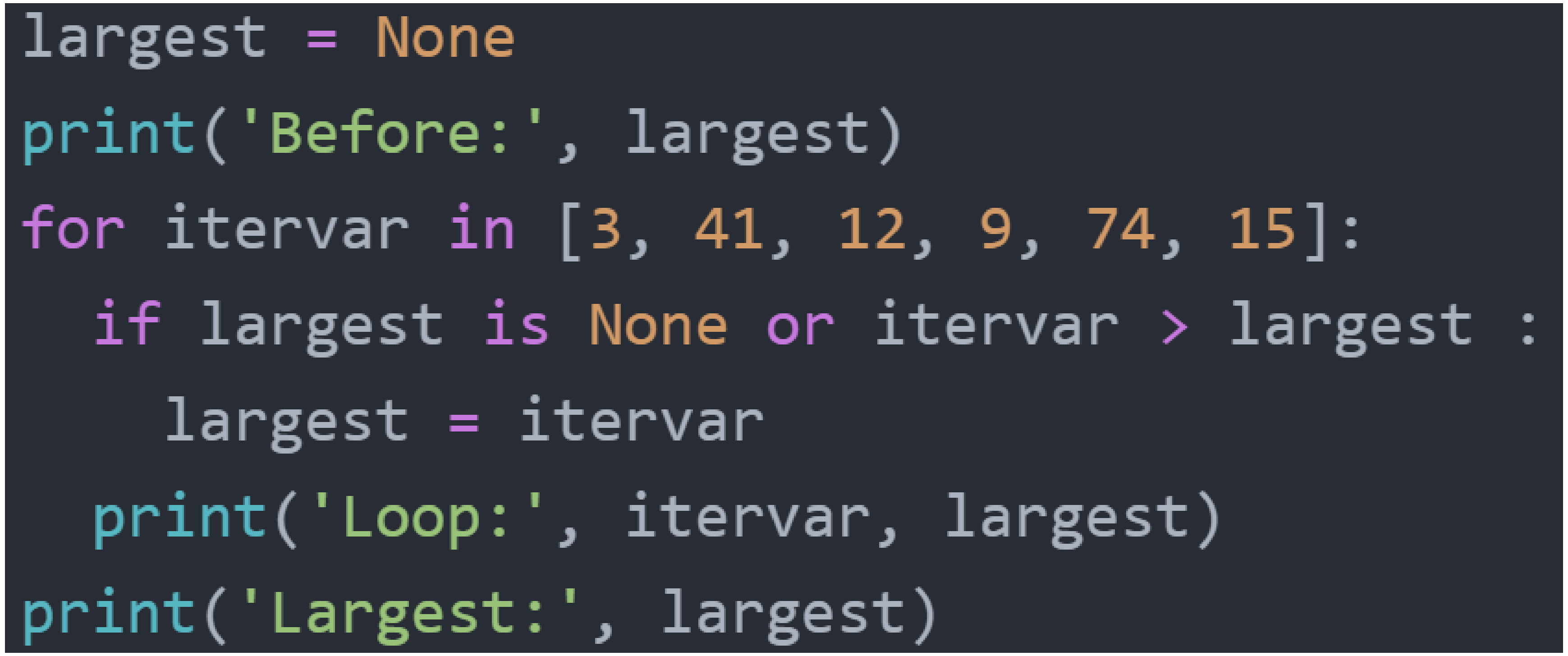
```
count = 0
for itervar in [3, 41, 12, 9, 74, 15]:
  count = count + 1
print('Count: ', count)
```

Counting and summing loops



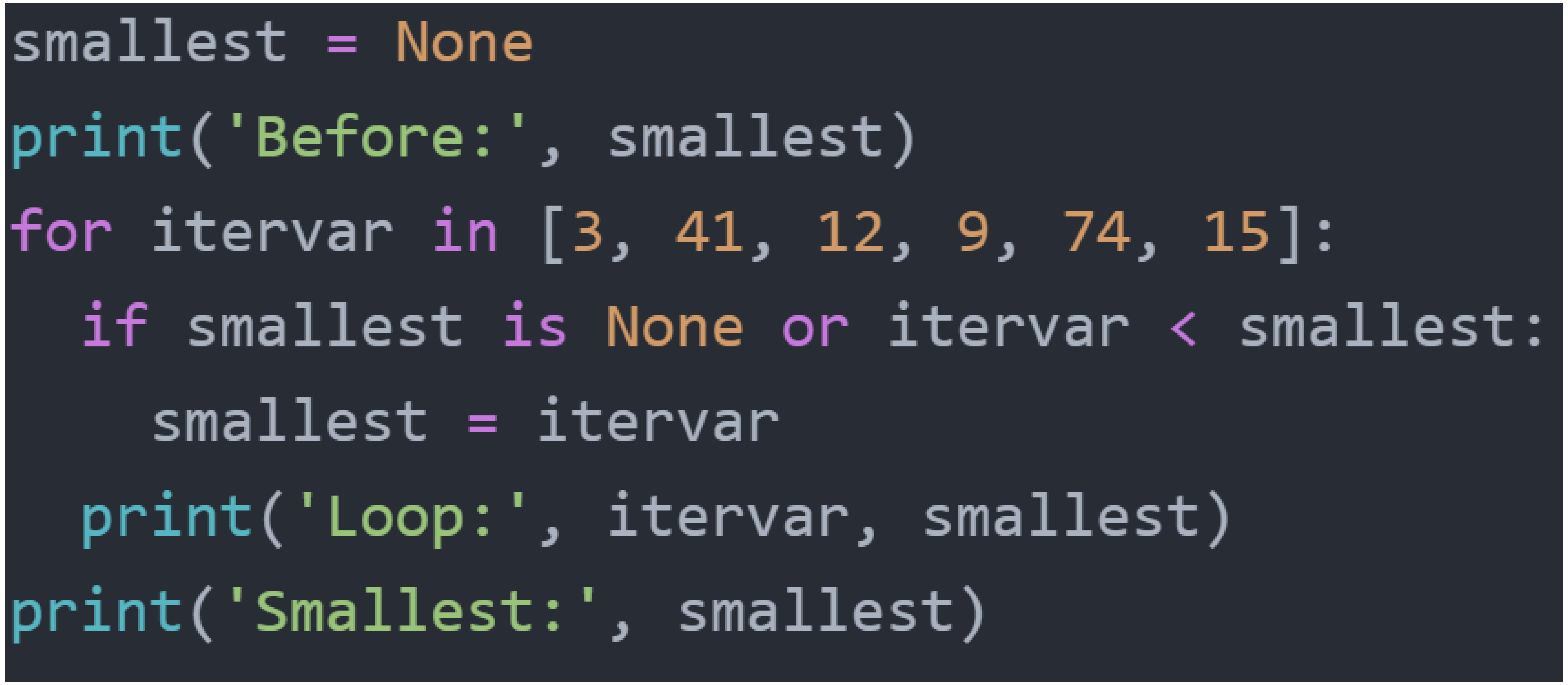
```
total = 0
for itervar in [3, 41, 12, 9, 74, 15]:
  total = total + itervar
  print('Total: ', total)
```

Maximum and minimum loops



```python
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)
```

Maximum and minimum loops



```python
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)
```

Maximum and minimum loops

- Again as in counting and summing, the built-in functions max() and min() make writing these exact loops unnecessary

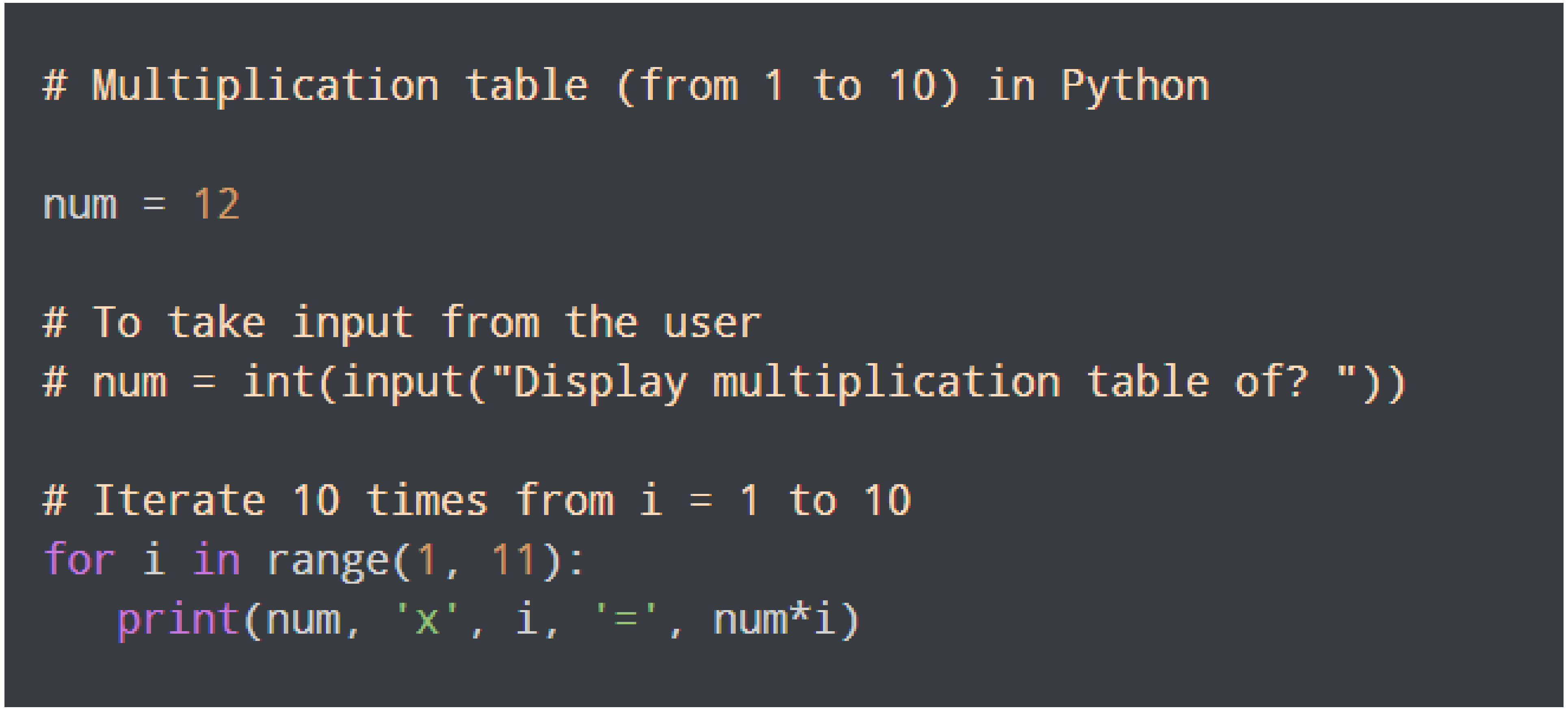
- max([1,2,3,4])
- min([1,2,3,4])

Activity – 9.1

- **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.**

Activity – 9.2

- **Write a program to display the multiplication table.**



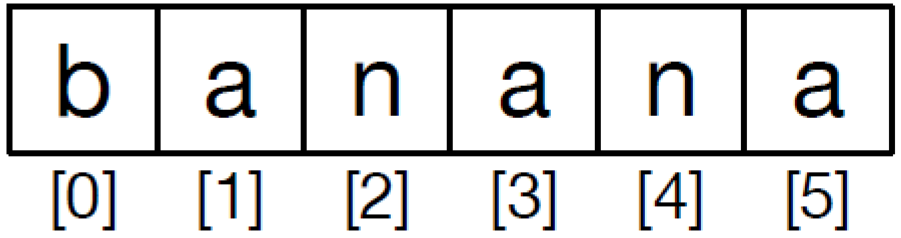
```python
# Multiplication table (from 1 to 10) in Python

num = 12

# To take input from the user
# num = int(input("Display multiplication table of? "))

# Iterate 10 times from i = 1 to 10
for i in range(1, 11):
   print(num, 'x', i, '=', num*i)
```

Strings

- 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]
```

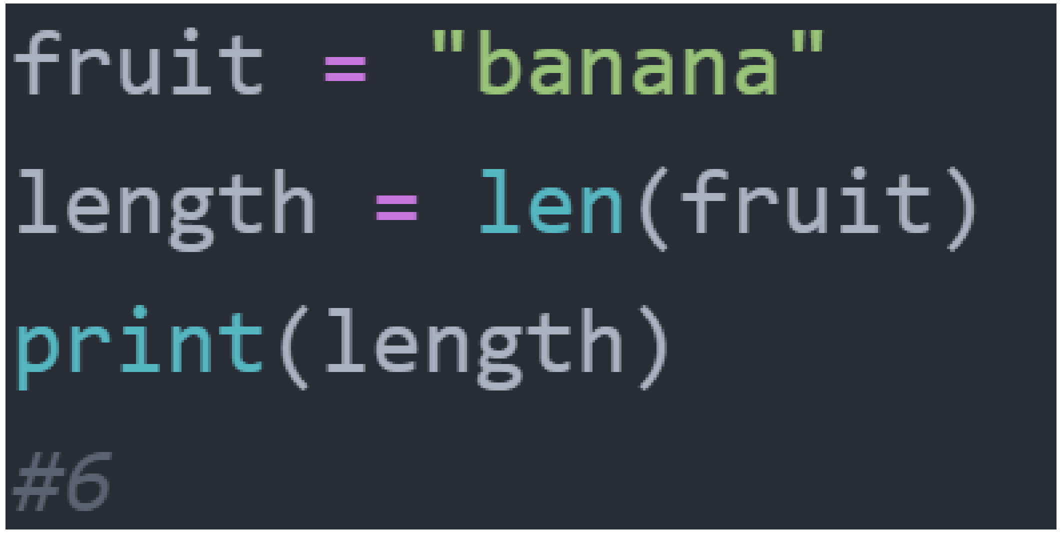
String Index



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

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

Getting the length of a string using len

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

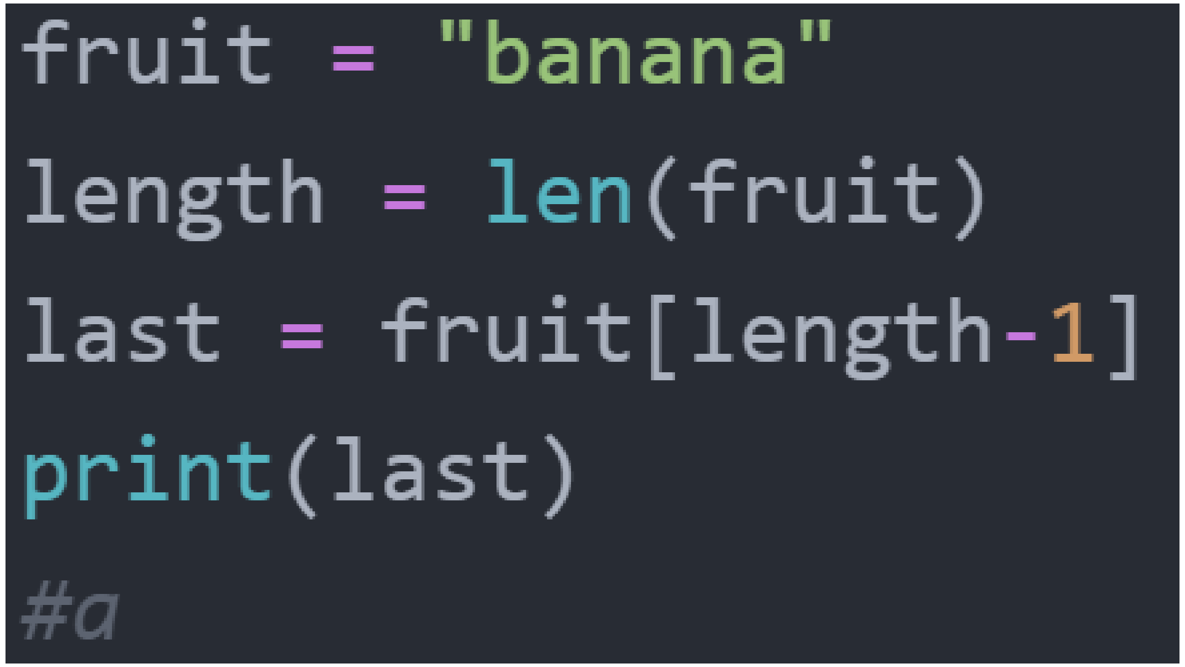


```
fruit = "banana"
length = len(fruit)
print(length)
#6
```

String index out of range

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

Getting last element of string

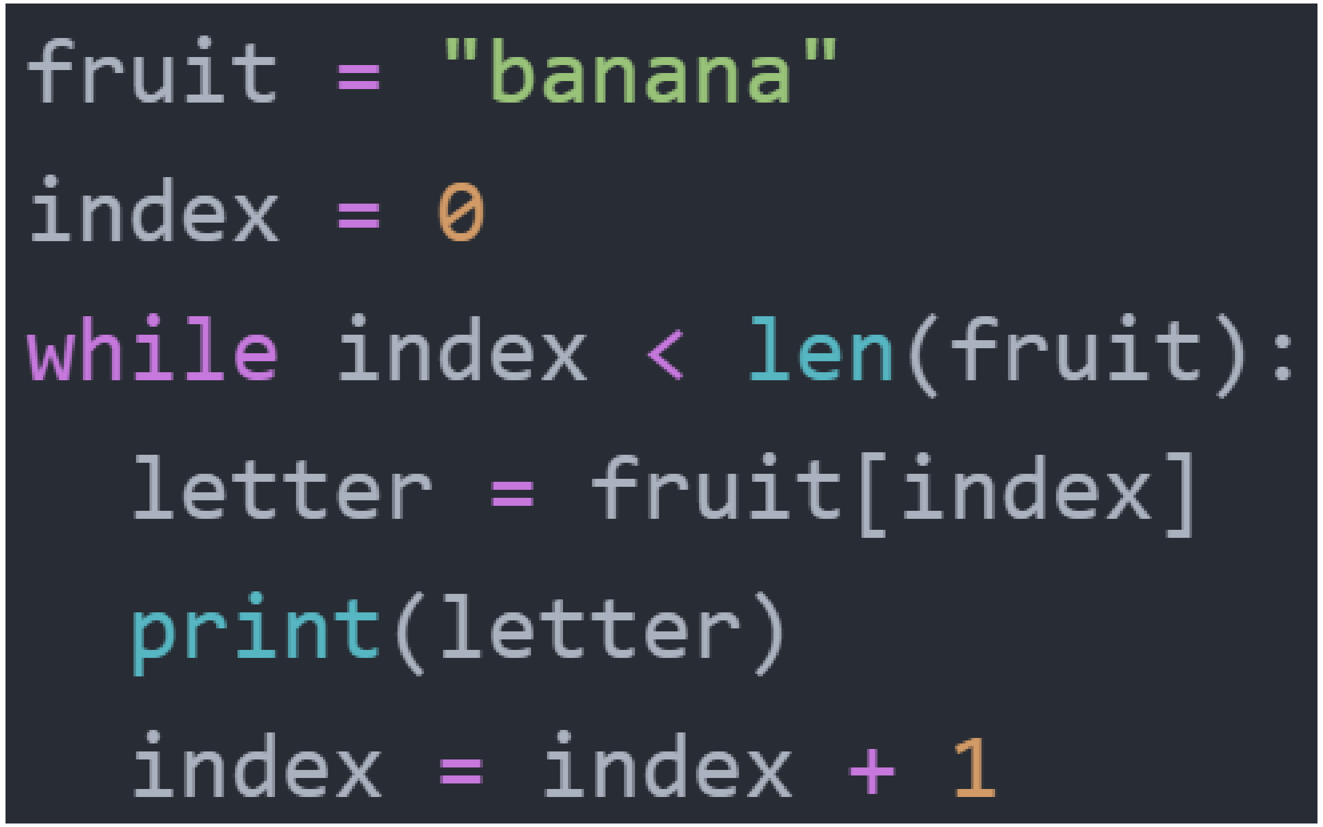


```
fruit = "banana"
length = len(fruit)
last = fruit[length-1]
print(last)
#a
```

Negative indices for String

- Alternatively, you can use negative indices, which count backward from the end of the string
- The expression fruit[-1] yields the last letter, fruit[-2] yields the second to last, and so on

Traversal through a string with a loop



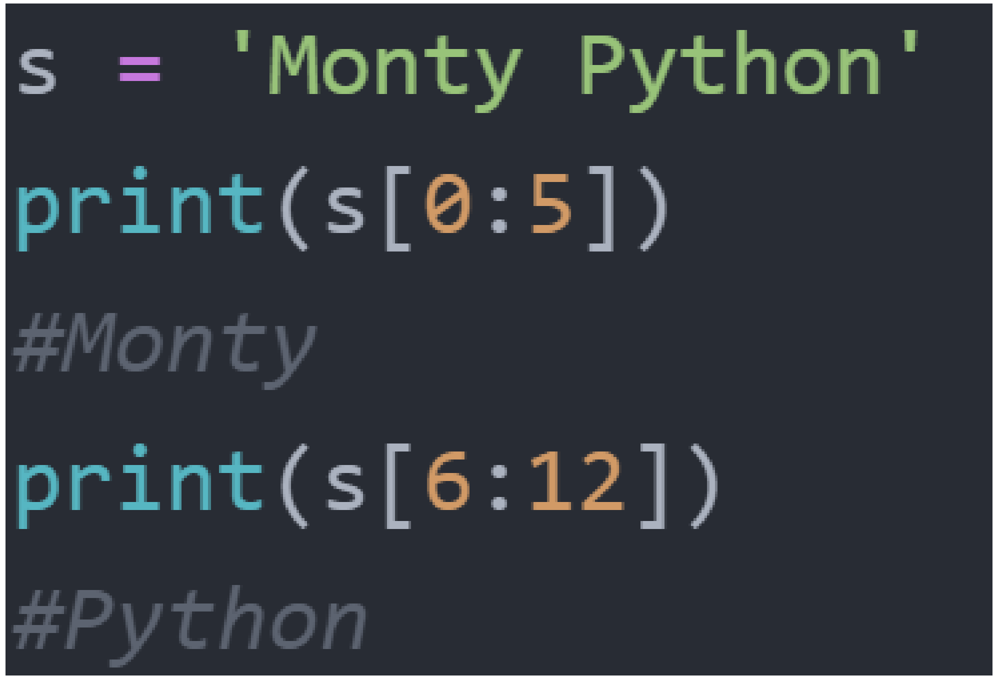
```python
fruit = "banana"
index = 0
while index < len(fruit):
  letter = fruit[index]
  print(letter)
  index = index + 1
```

Activity - 10

- **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 slices

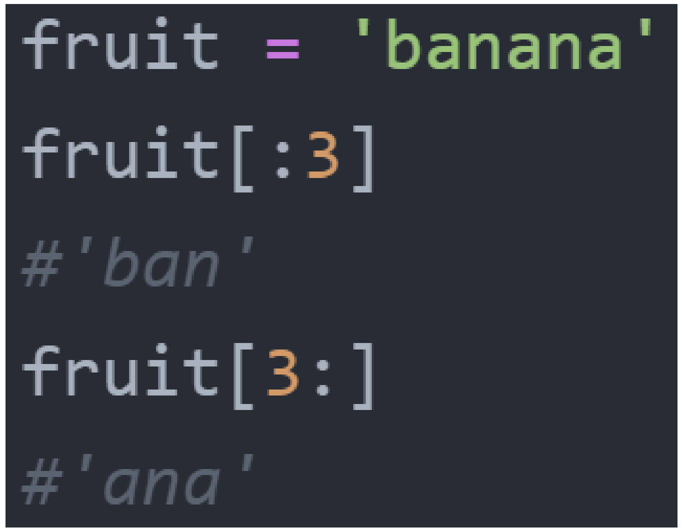
- A segment of a string is called a slice. Selecting a slice is similar to selecting a character:



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

String slices

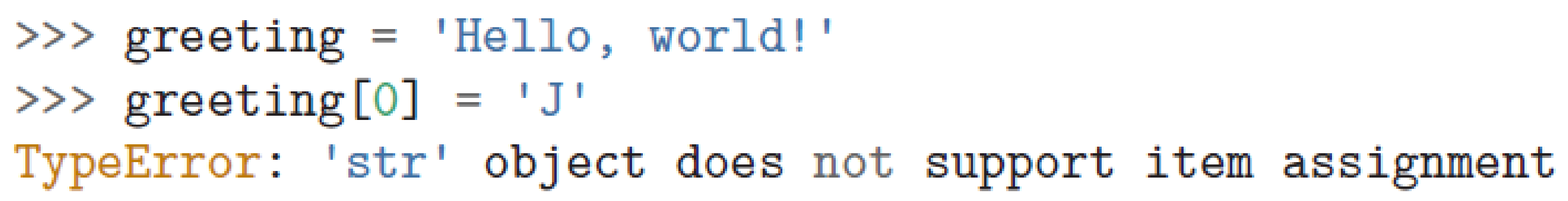
- If you omit the first index (before the colon), the slice starts at the beginning of the string
- If you omit the second index, the slice goes to the end of the string:



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

Strings are immutable

- It is tempting to use the operator on the left side of an assignment, with the intention of changing a character in a string



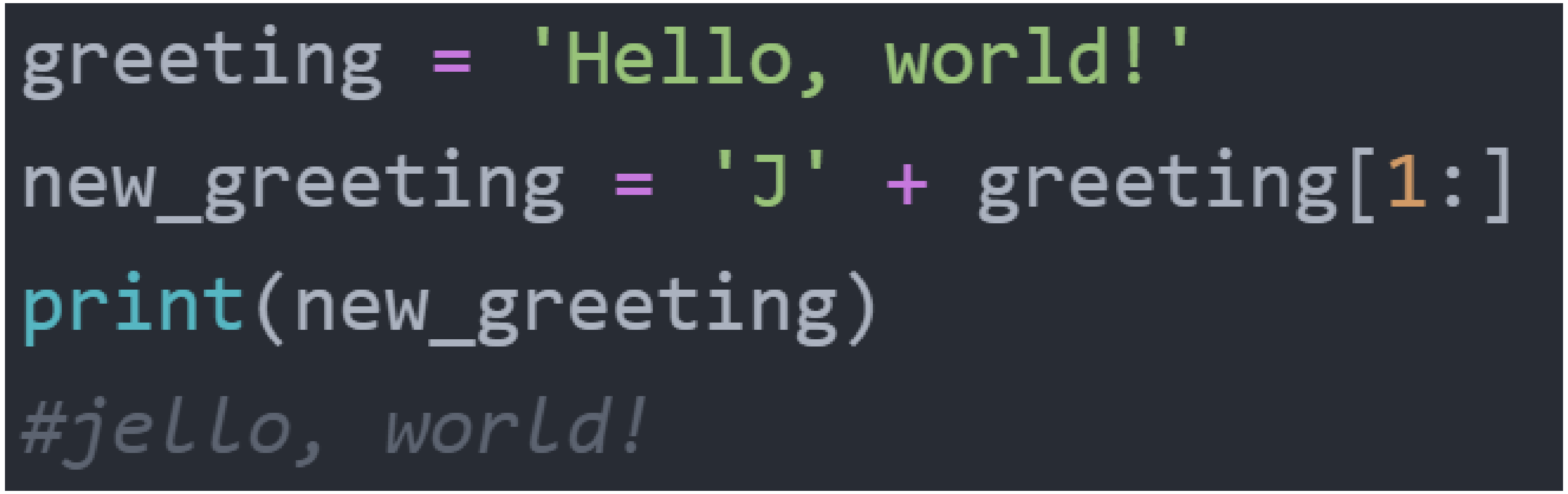
```
>>> greeting = 'Hello, world!'
>>> greeting[0] = 'J'
TypeError: 'str' object does not support item assignment
```

Strings are immutable

- The reason for the error is that strings are immutable, which means you can’t change an existing string

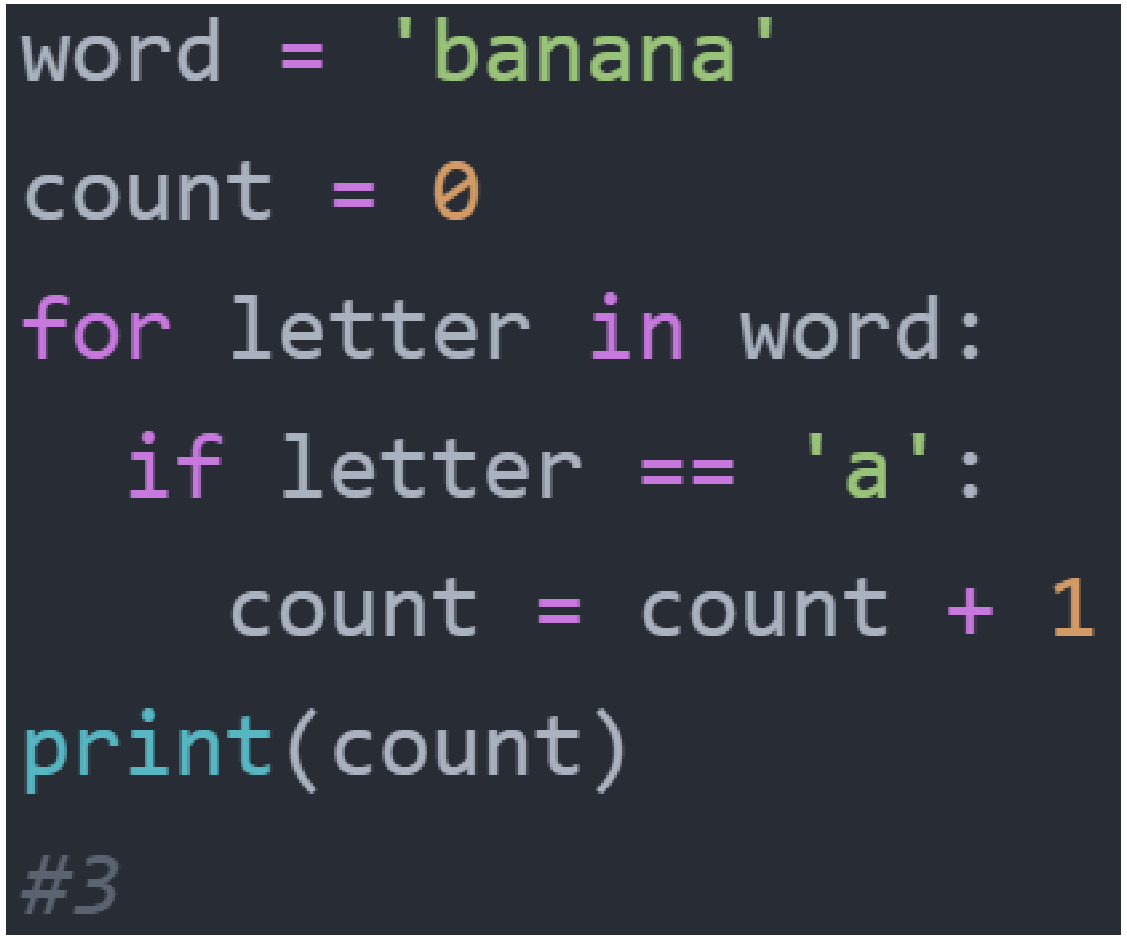
- The best you can do is create a new string that is a variation on the original:

Strings are immutable



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

Looping and counting



```
word = 'banana'
count = 0
for letter in word:
  if letter == 'a':
    count = count + 1
print(count)
#3
```

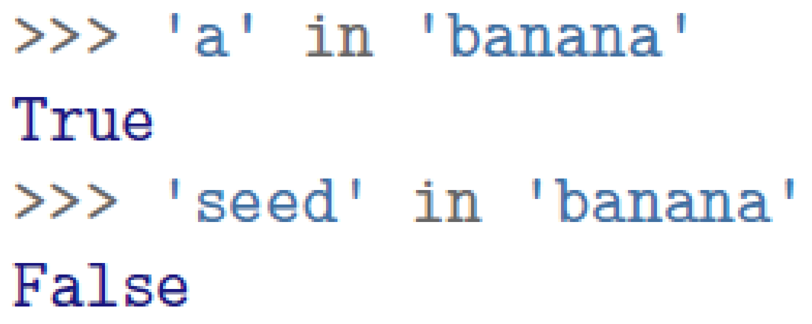
Activity - 11

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

```
def countLetters(str, ch):
    fruit = str
    count = 0
    for char in fruit:
        if char == ch:
            count = count + 1
    print(count)

countLetters("banana", "a")
```

The in operator

- The word **in** is a boolean operator that takes two strings and returns True if the first appears as a substring in the second:



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

Python Strings and Character Data

```
s = 'foo'
t = 'bar'
print('barf' in 2 * (s + t))
foobarfoobar
```

What is the output of the print() function call?

- True
- False

Explanation

- The + operator concatenates strings and the * operator creates multiple copies. The result of 2 * (s + t) is 'foobarfoobar', which does contain the string 'barf'.

Python Strings and Character Data

What is the result of this statement?

print(ord('foo’))

What is the output of the print() function call?

- It raises an exception
- 102 111 111
- 324
- 102

Explanation

- The ord() function returns the integer value for a given character. But you can only specify a single character (a string of length 1):

- >>> print(ord('f'))
- 102

Python Strings and Character Data

What is the slice expression that gives every third character of string s, starting with the last character and proceeding backward to the first?

- s = ‘1234567890abcdefg’

Explanation

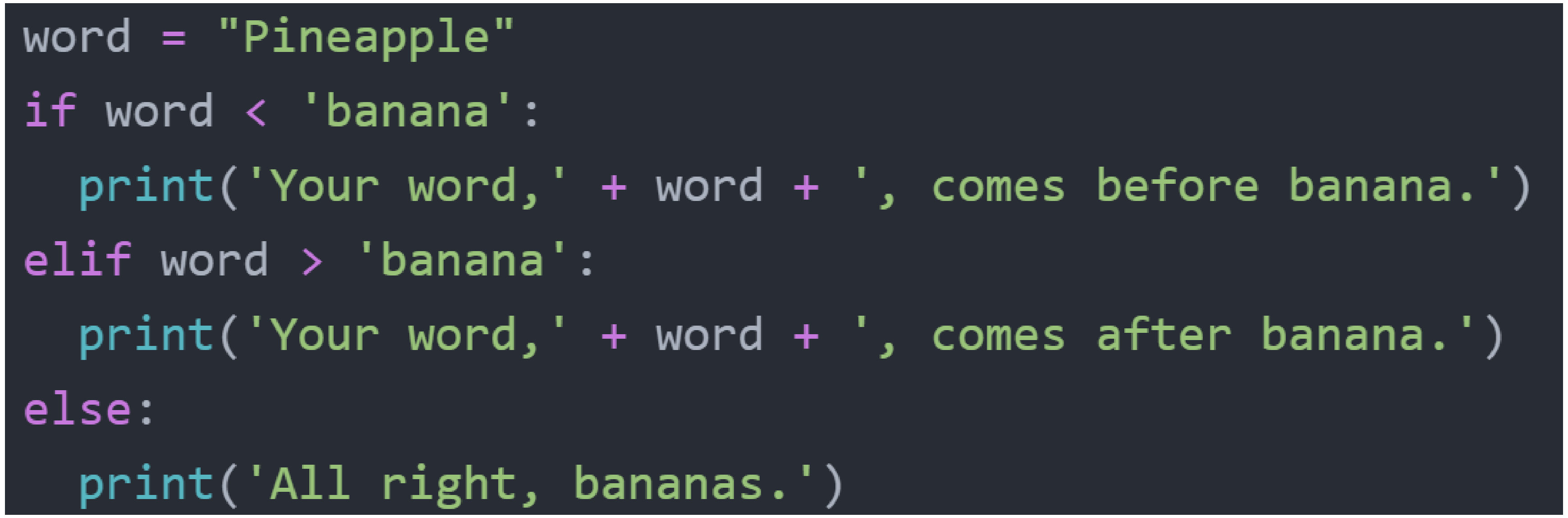
- s[::-3]

String comparison

- The comparison operators work on strings. To see if two strings are equal:

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

String Comparison



```
word = "Pineapple"
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.')
```

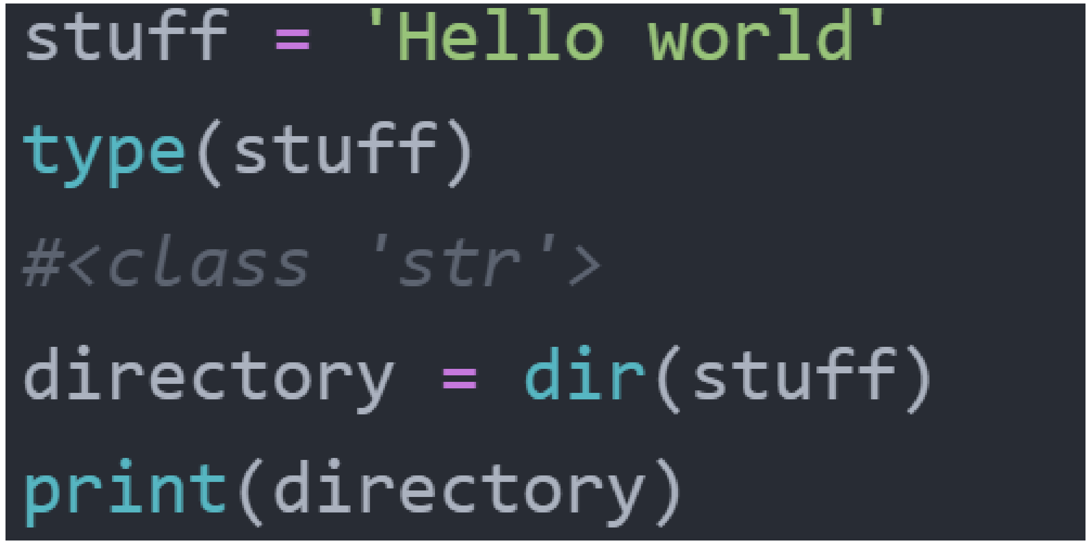
String methods

- 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

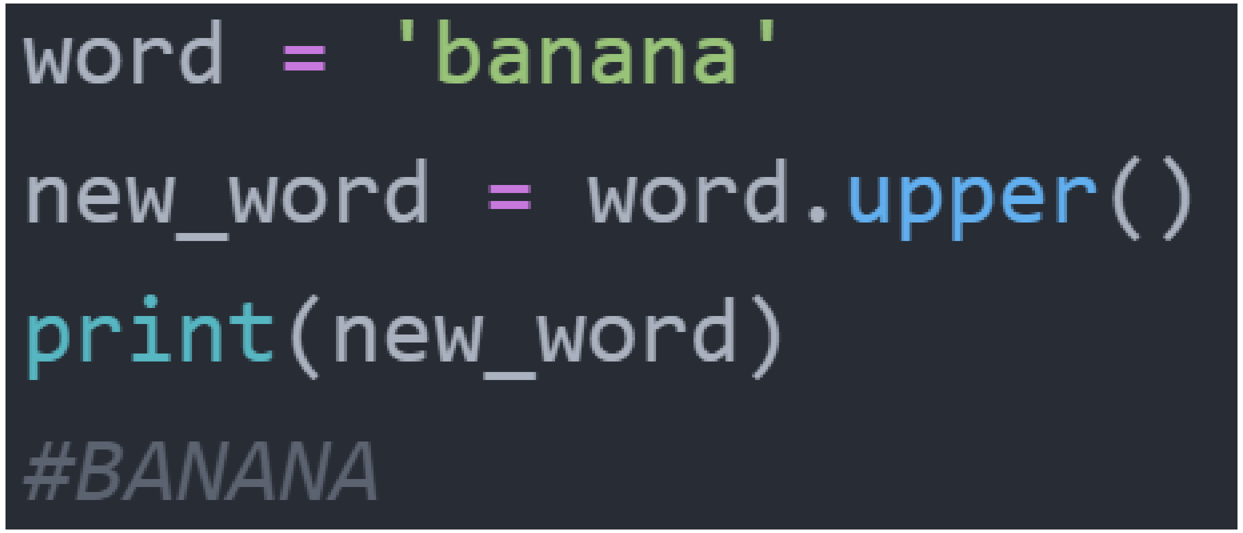
String methods



```python
stuff = 'Hello world'
type(stuff)
#<class 'str'>
directory = dir(stuff)
print(directory)
```

```
['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']
```

Upper()



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

Activity – 12

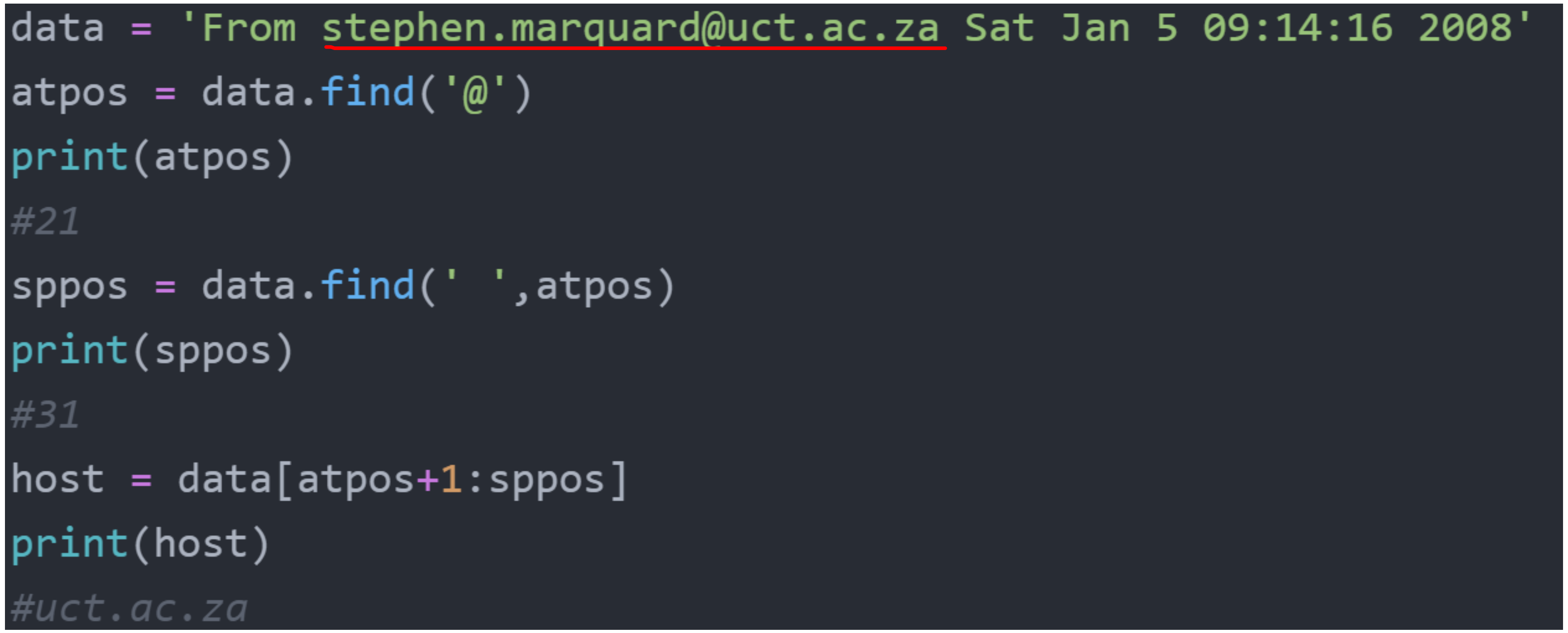
- Use the following build-in functions
  - find()
  - strip()
  - startswith()
  - lower()

Parsing strings

- Often, we want to look into a string and find a substring. For example if we were presented a series of lines formatted as follows

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

Parsing Strings


```
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
```

Activity – 12a

- Write a Python program to convert a given string into a list of words.
- Write a Python program to lowercase first n characters in a string.
- Write a Python program to count and display the vowels of a given text.
- Write a Python program to remove the characters which have odd index values of a given string.