Bit Project

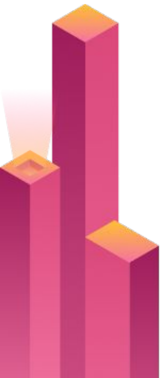
Introduction to Python

Module 1: Part 2

Bit Project

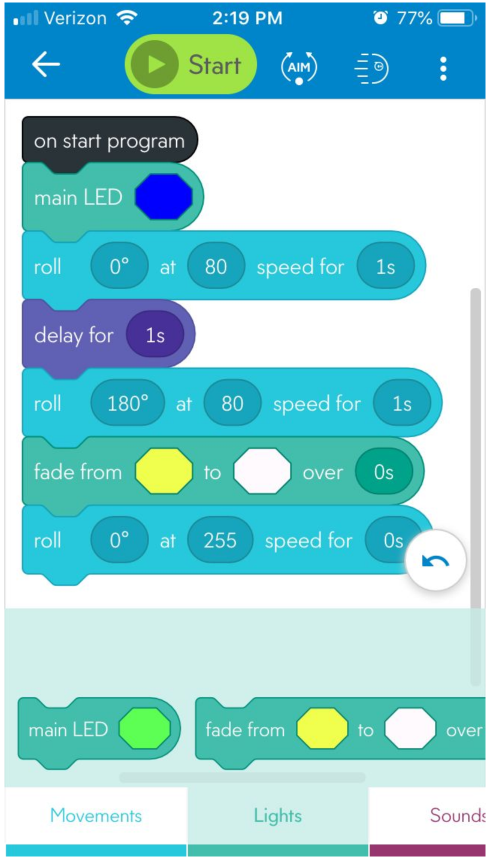
Introduction to Python

Let’s get started!



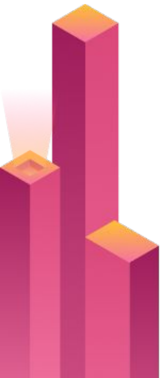
Bit Project

Overview



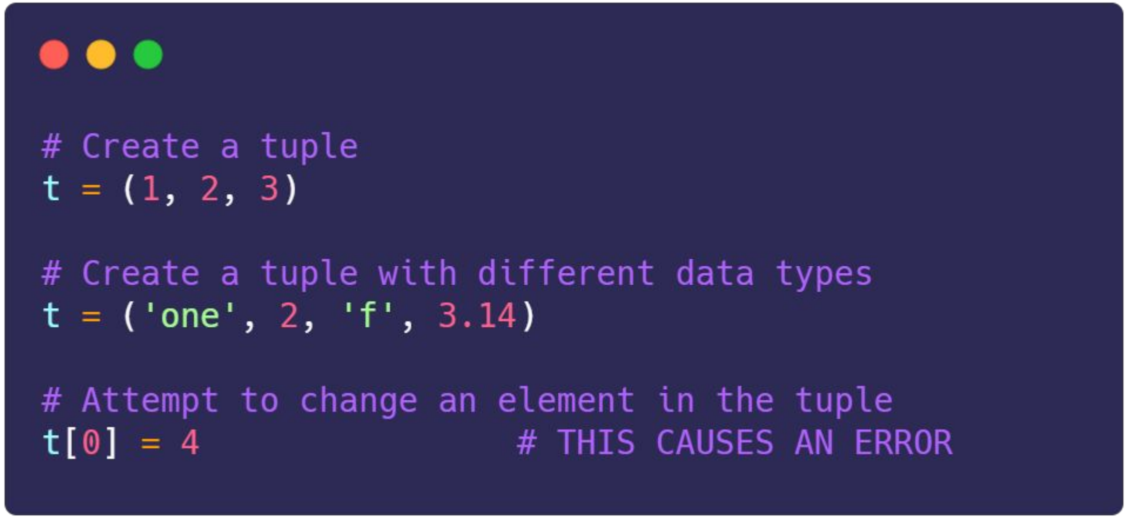
Bit Project

Tuples



Creating Tuples

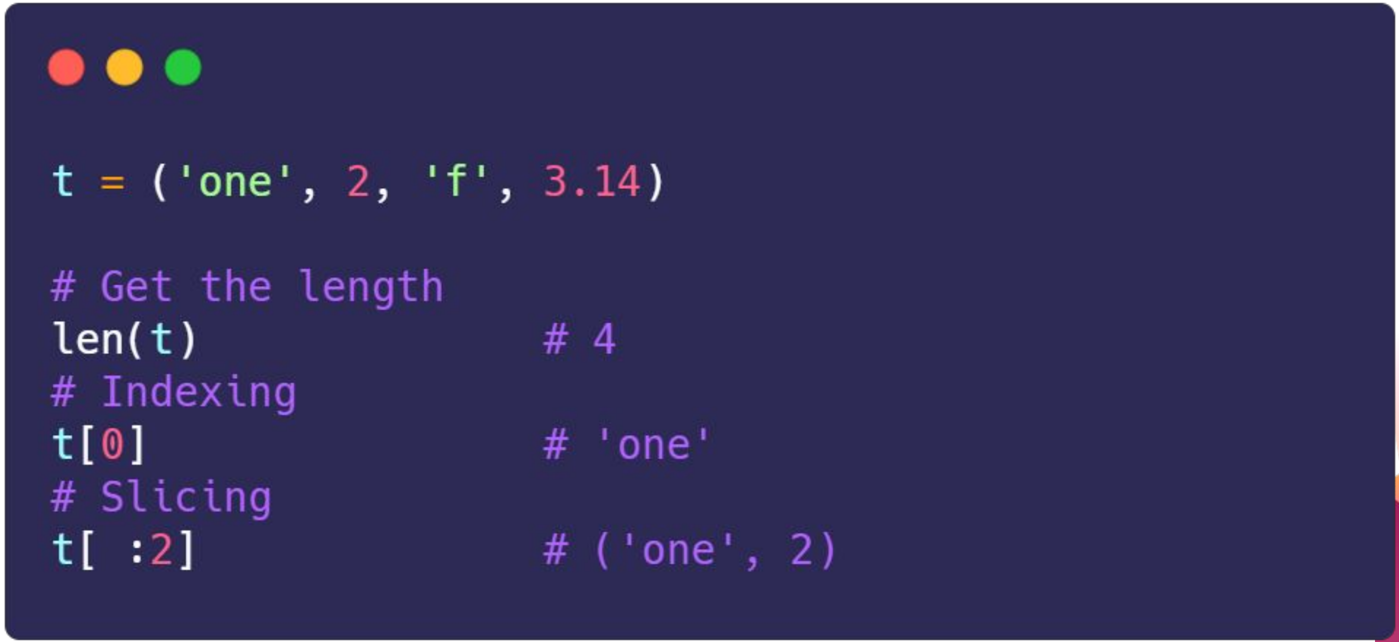
- **Tuples** are a data type that are similar to **lists**, but are **immutable**, and **CANNOT be changed after creation**
  - `Format:     ( a, b, c )`



```
# Create a tuple
t = (1, 2, 3)

# Create a tuple with different data types
t = ('one', 2, 'f', 3.14)

# Attempt to change an element in the tuple
t[0] = 4                # THIS CAUSES AN ERROR
```

Tuple Properties

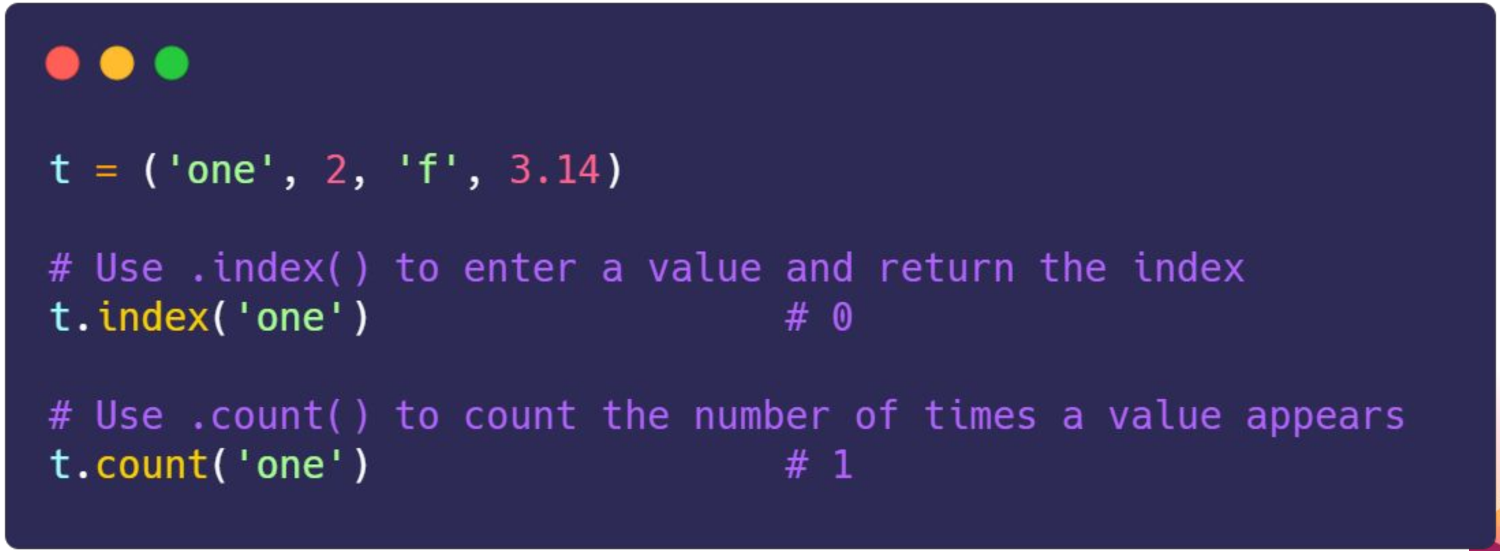
- **Tuples** have many of the same properties that **lists** have



```
t = ('one', 2, 'f', 3.14)

# Get the length
len(t)                  # 4
# Indexing
t[0]                    # 'one'
# Slicing
t[ :2]                  # ('one', 2)
```

Bit Project

Built-In Tuple Methods

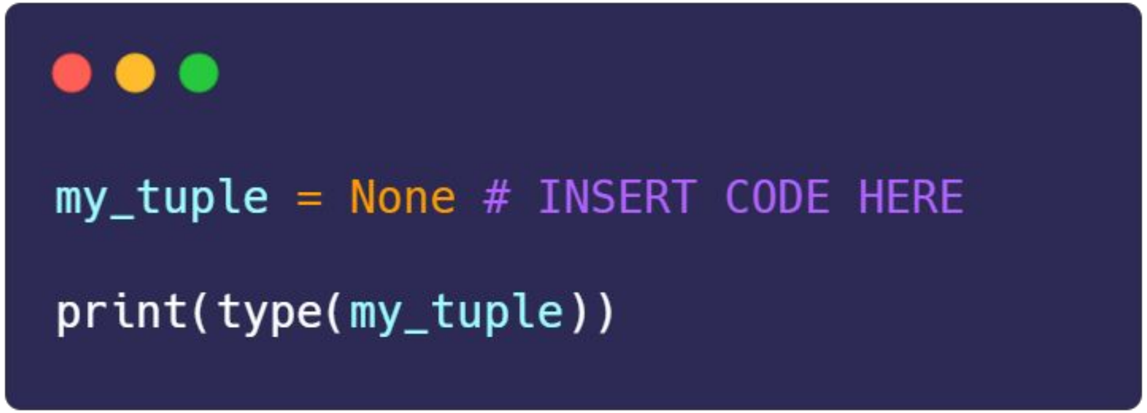


```python
t = ('one', 2, 'f', 3.14)

# Use .index() to enter a value and return the index
t.index('one')                         # 0

# Use .count() to count the number of times a value appears
t.count('one')                         # 1
```

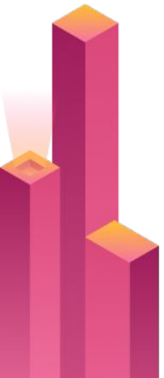
Bit Project

Now Try This

Create a tuple and verify with the below code:


```
my_tuple = None # INSERT CODE HERE

print(type(my_tuple))
```

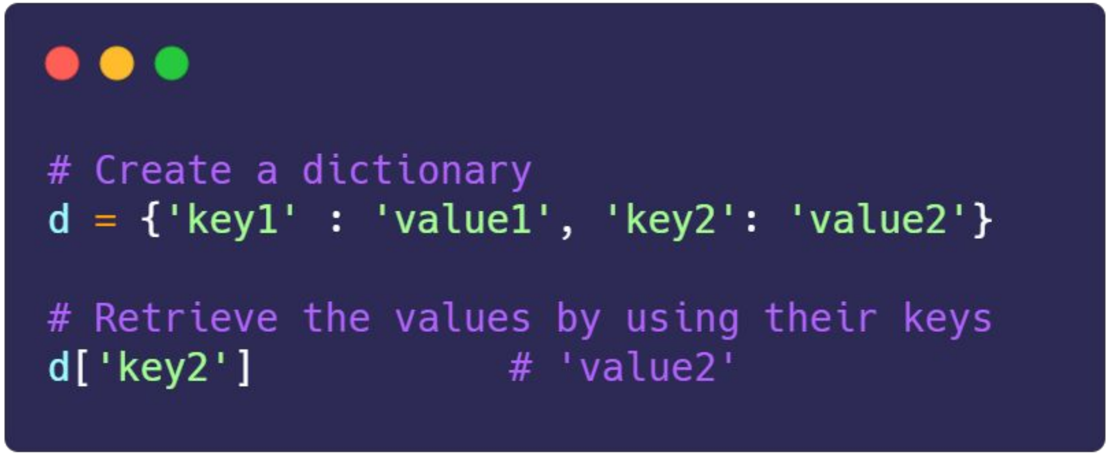
Dictionaries



Bit Project

Creating Dictionaries

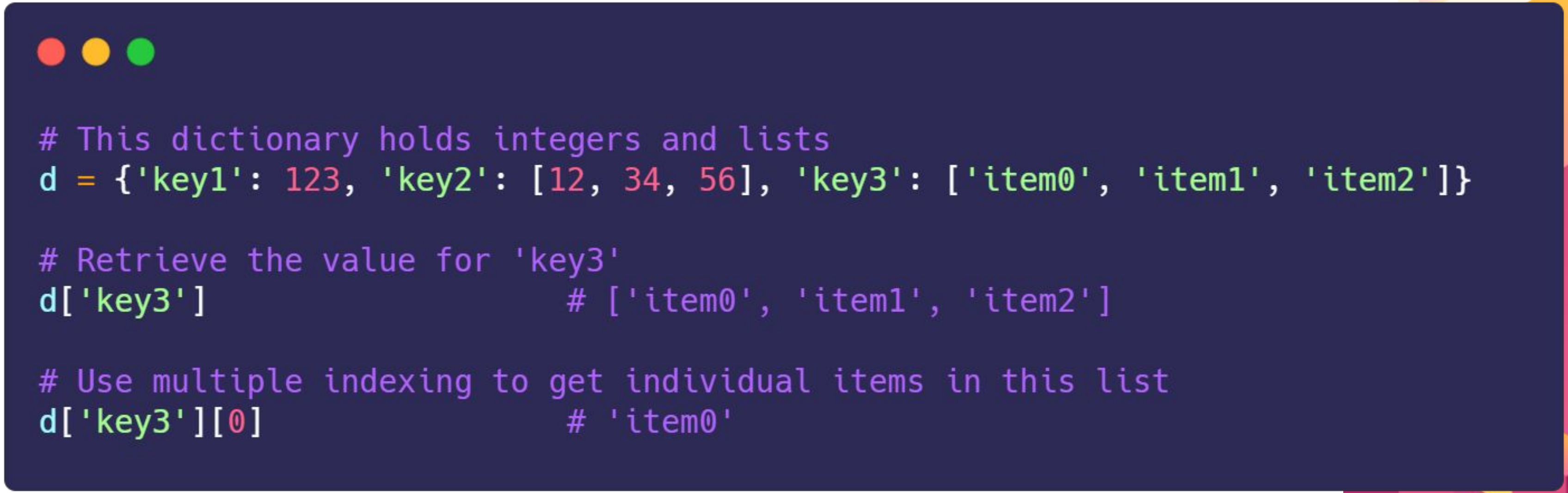
- **Dictionaries** are a data type that store a **mapping** of **key-value pairs**
    - Format: `{ key1 : value1, key2 : value2, … }`



```
# Create a dictionary
d = {'key1' : 'value1', 'key2': 'value2'}

# Retrieve the values by using their keys
d['key2']             # 'value2'
```

Indexing Dictionaries

- **Dictionaries** can store different data types as their values, creating a more complex structure

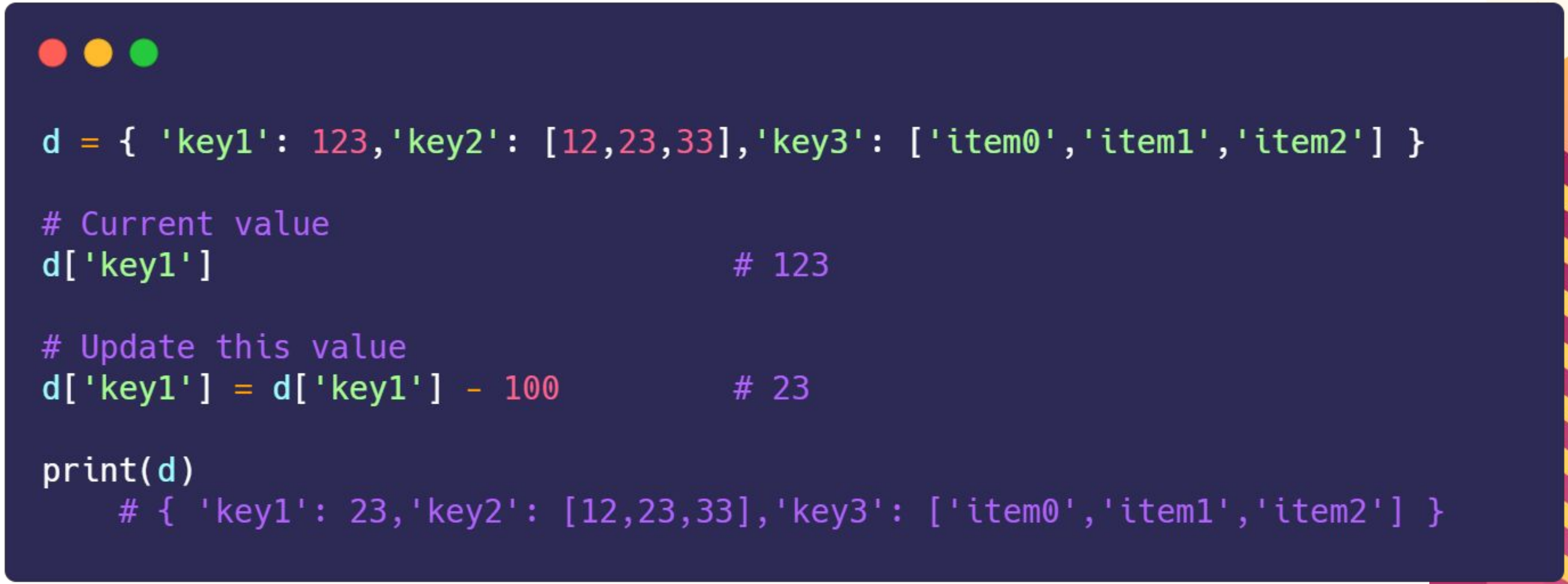


```python
# This dictionary holds integers and lists
d = {'key1': 123, 'key2': [12, 34, 56], 'key3': ['item0', 'item1', 'item2']}

# Retrieve the value for 'key3'
d['key3']                           # ['item0', 'item1', 'item2']

# Use multiple indexing to get individual items in this list
d['key3'][0]                        # 'item0'
```

Bit Project

Dictionary Mutability

- We can change values in dictionaries

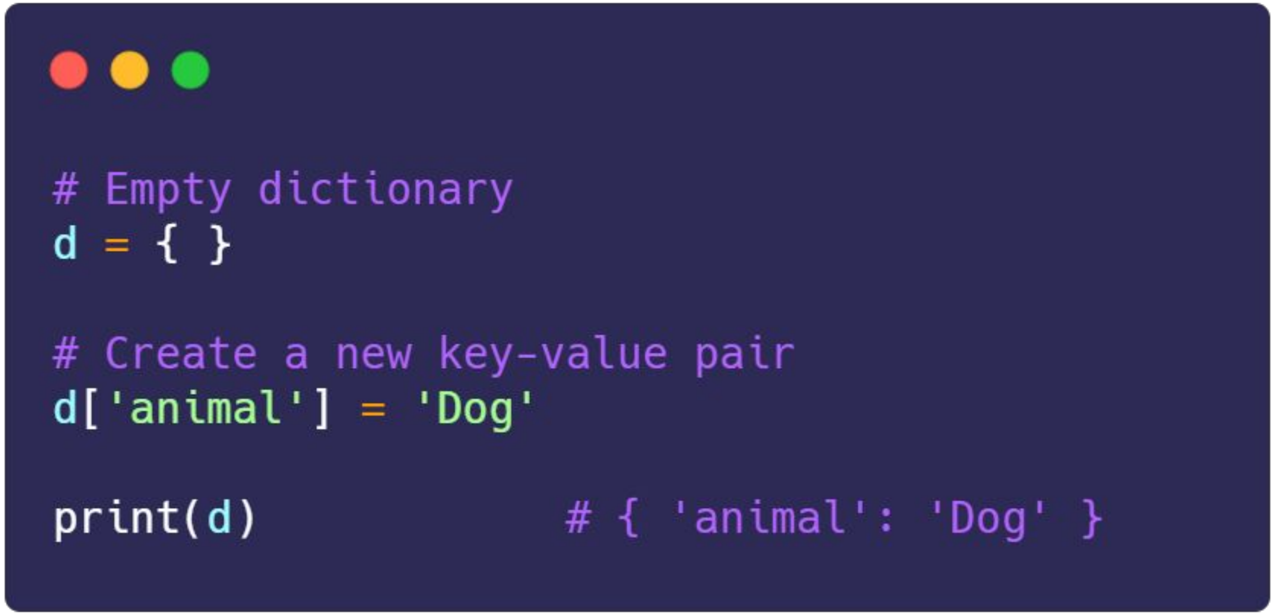


```python
d = { 'key1': 123,'key2': [12,23,33],'key3': ['item0','item1','item2'] }

# Current value
d['key1']                           # 123

# Update this value
d['key1'] = d['key1'] - 100         # 23

print(d)
    # { 'key1': 23,'key2': [12,23,33],'key3': ['item0','item1','item2'] }
```

Bit Project

Empty Dictionaries

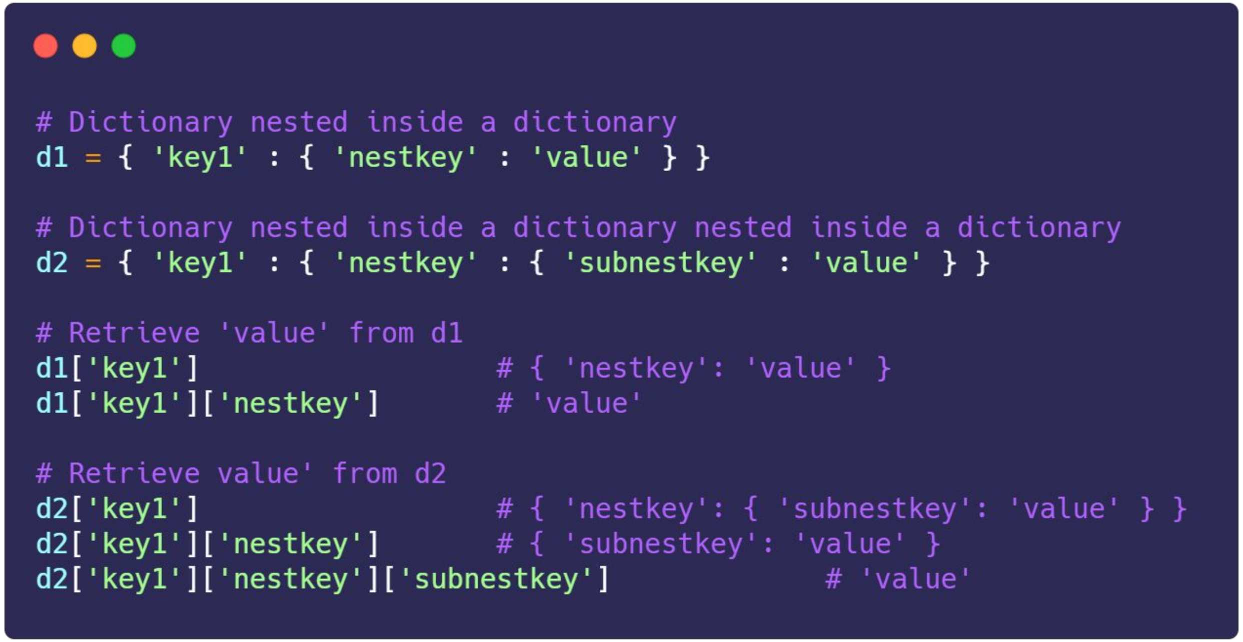
- We can initialize **empty dictionaries and add values later**



```
# Empty dictionary
d = { }

# Create a new key-value pair
d['animal'] = 'Dog'

print(d)                # { 'animal': 'Dog' }
```

Nested Dictionaries

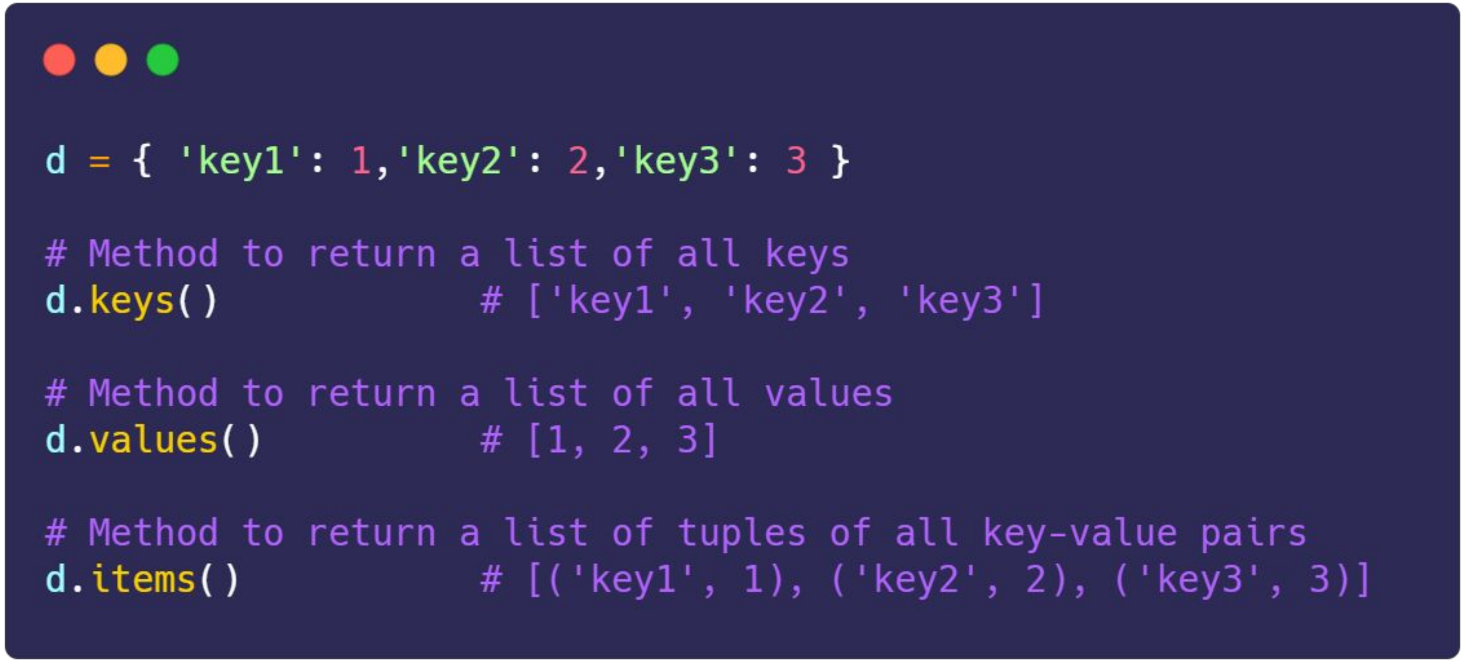
- We can initialize **empty dictionaries and add values later**



```
# Dictionary nested inside a dictionary
d1 = { 'key1' : { 'nestkey' : 'value' } }

# Dictionary nested inside a dictionary nested inside a dictionary
d2 = { 'key1' : { 'nestkey' : { 'subnestkey' : 'value' } }

# Retrieve 'value' from d1
d1['key1']                  # { 'nestkey': 'value' }
d1['key1']['nestkey']       # 'value'

# Retrieve value' from d2
d2['key1']                  # { 'nestkey': { 'subnestkey': 'value' } }
d2['key1']['nestkey']       # { 'subnestkey': 'value' }
d2['key1']['nestkey']['subnestkey']                # 'value'
```

Bit Project

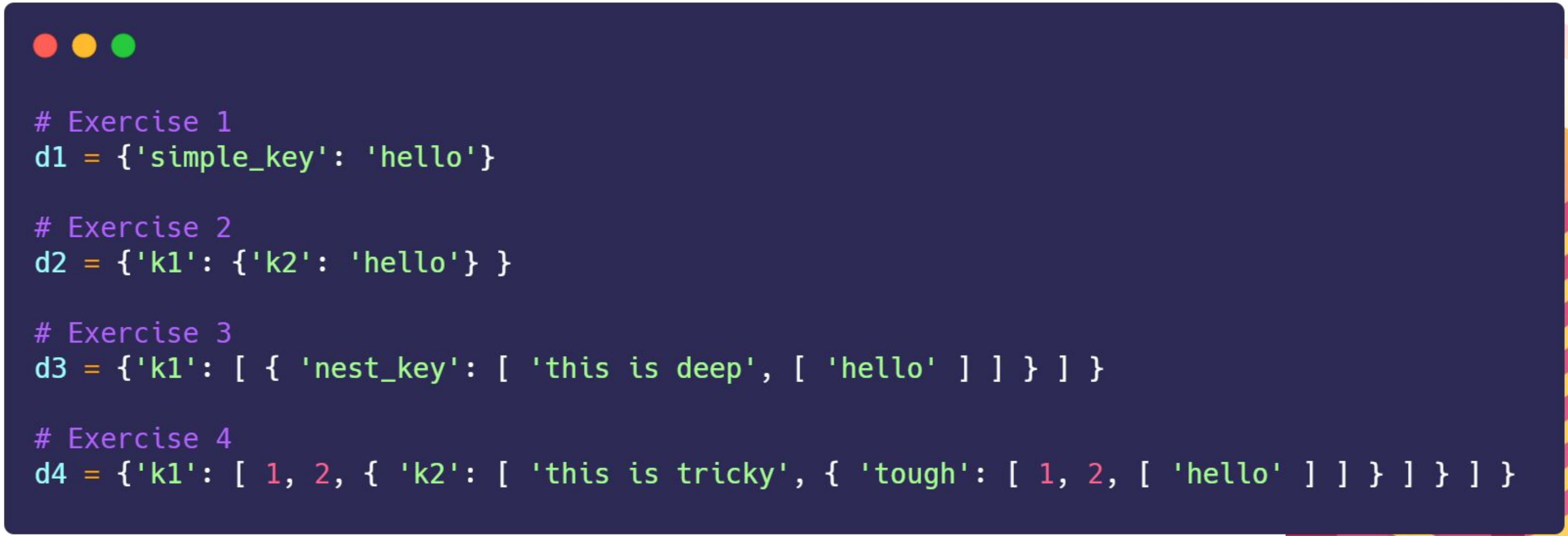
Built-In Dictionary Methods



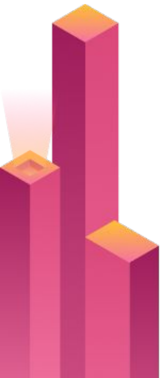
```
d = { 'key1': 1,'key2': 2,'key3': 3 }

# Method to return a list of all keys
d.keys()                # ['key1', 'key2', 'key3']

# Method to return a list of all values
d.values()              # [1, 2, 3]

# Method to return a list of tuples of all key-value pairs
d.items()               # [('key1', 1), ('key2', 2), ('key3', 3)]
```

Now Try This

Grab ‘hello’ from the following dictionaries using keys and indexing.



```python
# Exercise 1
d1 = {'simple_key': 'hello'}

# Exercise 2
d2 = {'k1': {'k2': 'hello'} }

# Exercise 3
d3 = {'k1': [ { 'nest_key': [ 'this is deep', [ 'hello' ] ] } ] }

# Exercise 4
d4 = {'k1': [ 1, 2, { 'k2': [ 'this is tricky', { 'tough': [ 1, 2, [ 'hello' ] ] } ] } ] }
```



Loops

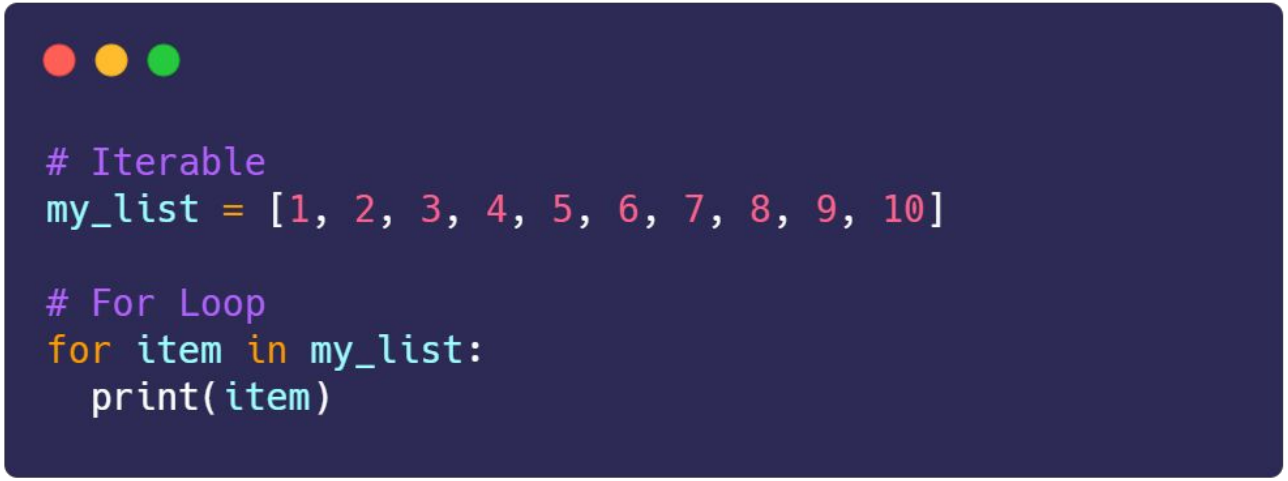
Bit Project

Types of Loops

- Loops are used to run specific lines of code multiple times
- Types of Loops:
  - For Loop
  - While Loop

for Loops (1)

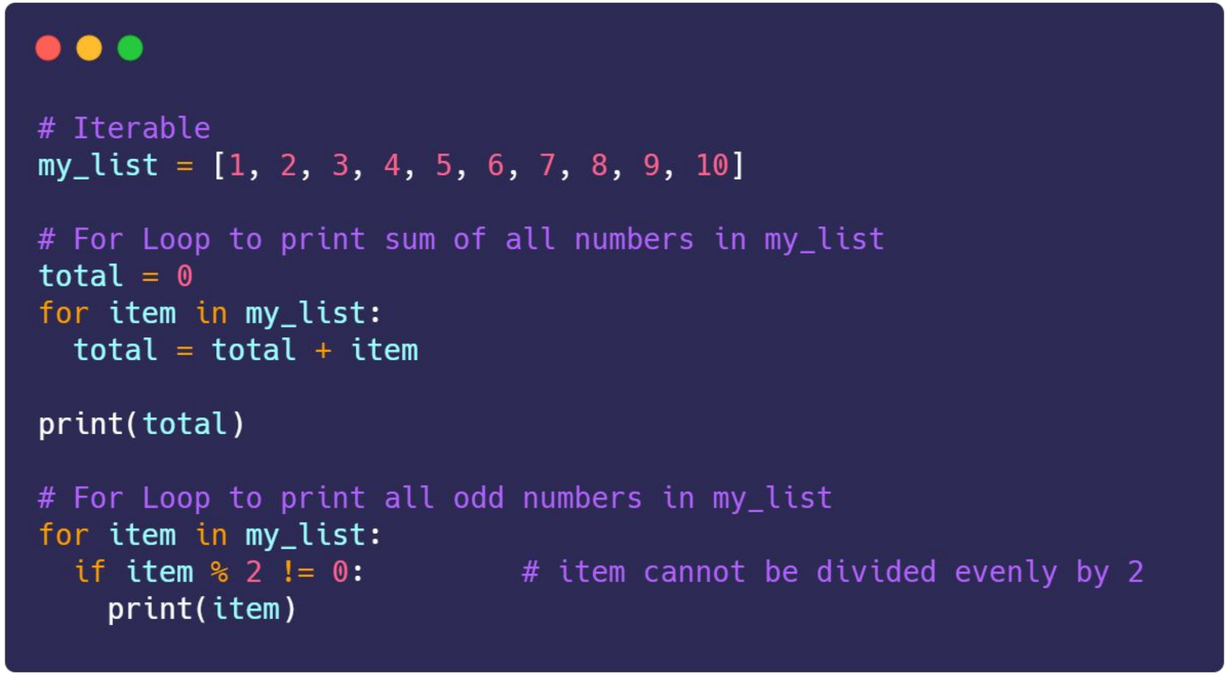
- **For Loops** perform an action for every element in a sequence
    - Format: “For every element in this iterable, do this”



```
# Iterable
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# For Loop
for item in my_list:
  print(item)
```

Bit Project

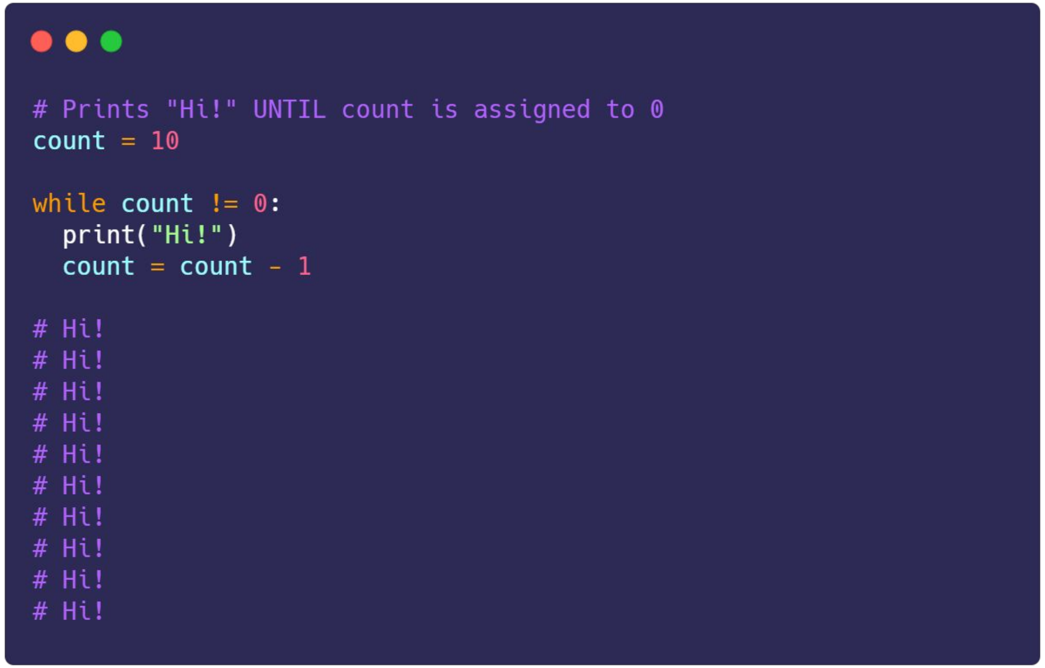
for Loops (2)



```python
# Iterable
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# For Loop to print sum of all numbers in my_list
total = 0
for item in my_list:
  total = total + item

print(total)

# For Loop to print all odd numbers in my_list
for item in my_list:
  if item % 2 != 0:         # item cannot be divided evenly by 2
    print(item)
```

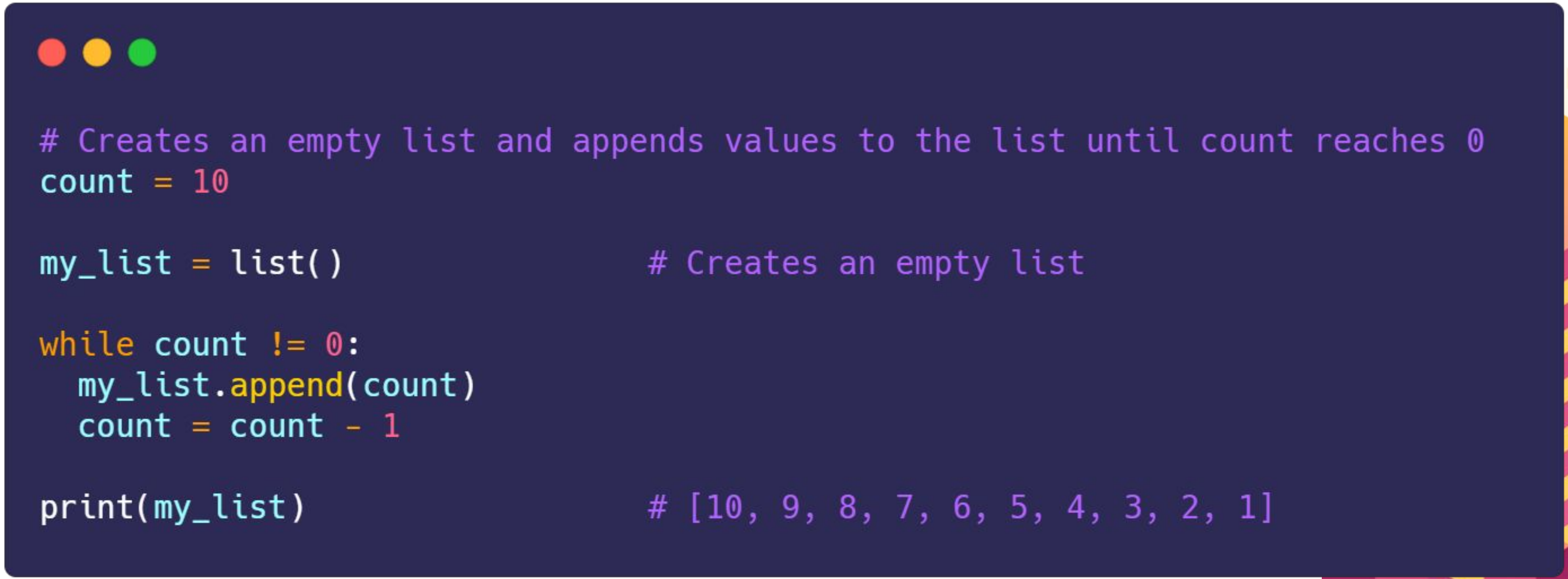
while Loops (1)

- **While loops** continue as long as a condition is met



```python
# Prints "Hi!" UNTIL count is assigned to 0
count = 10

while count != 0:
  print("Hi!")
  count = count - 1

# Hi!
# Hi!
# Hi!
# Hi!
# Hi!
# Hi!
# Hi!
# Hi!
# Hi!
# Hi!
```

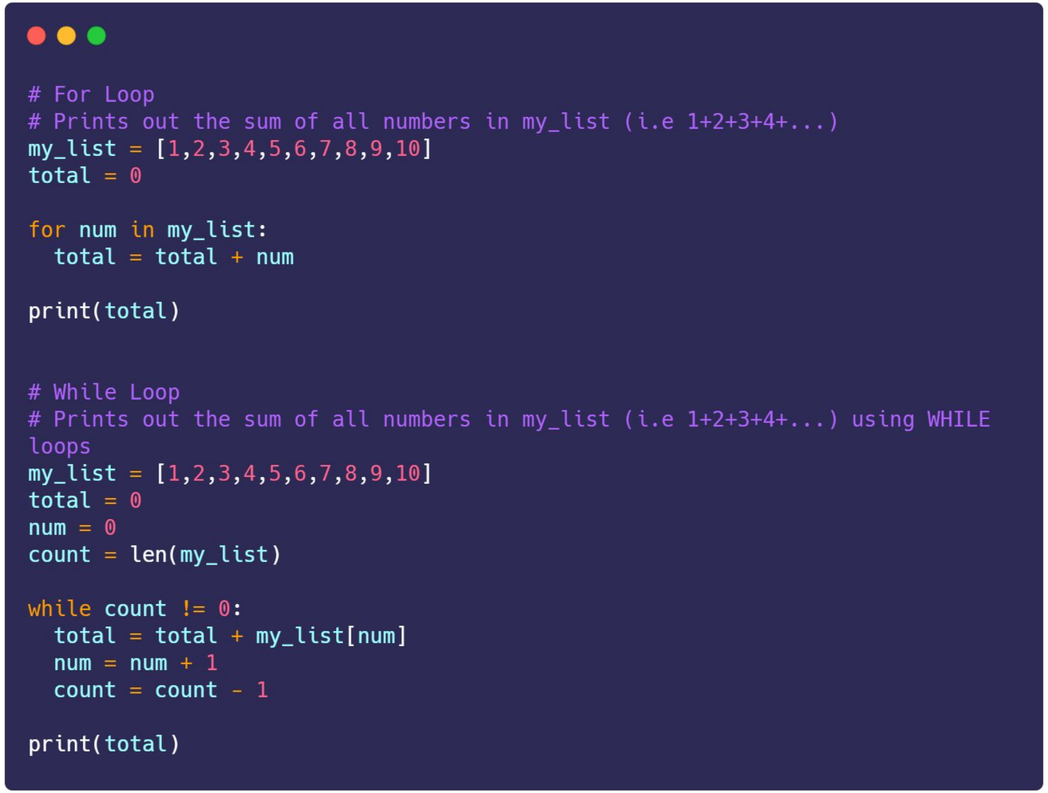
while Loops (2)



```python
# Creates an empty list and appends values to the list until count reaches 0
count = 10

my_list = list()                # Creates an empty list

while count != 0:
  my_list.append(count)
  count = count - 1

print(my_list)                  # [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Bit Project

for / while Loop Interchangeability

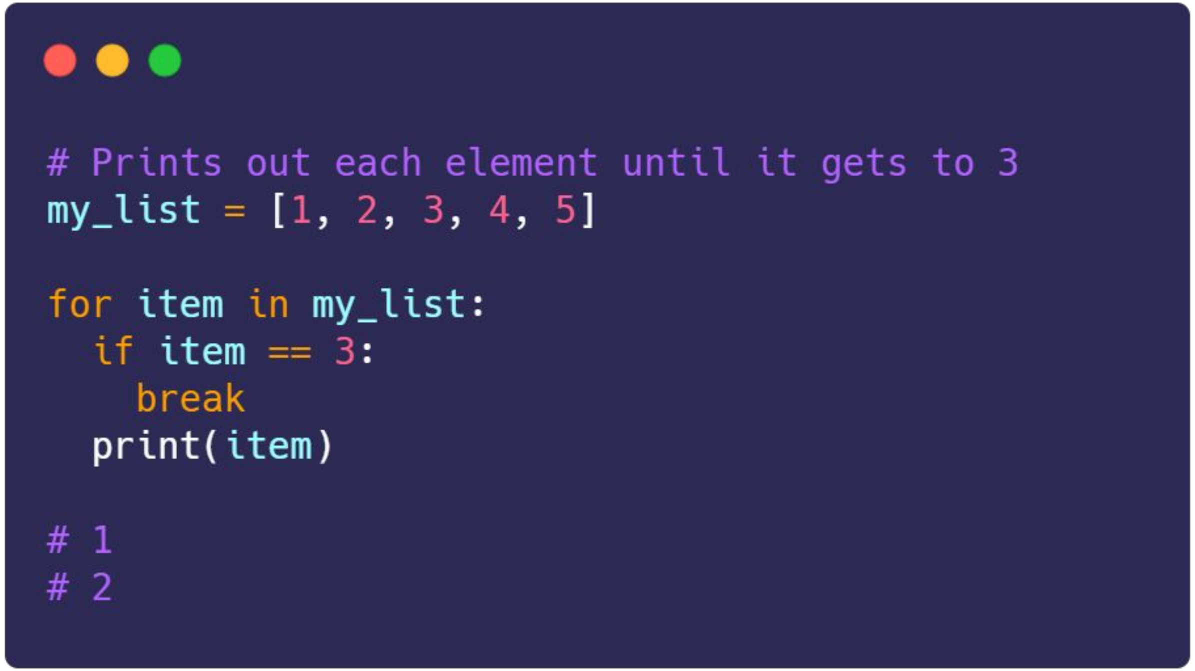


```
# For Loop
# Prints out the sum of all numbers in my_list (i.e 1+2+3+4+...)
my_list = [1,2,3,4,5,6,7,8,9,10]
total = 0

for num in my_list:
  total = total + num

print(total)


# While Loop
# Prints out the sum of all numbers in my_list (i.e 1+2+3+4+...) using WHILE
loops
my_list = [1,2,3,4,5,6,7,8,9,10]
total = 0
num = 0
count = len(my_list)

while count != 0:
  total = total + my_list[num]
  num = num + 1
  count = count - 1

print(total)
```

Bit Project

break

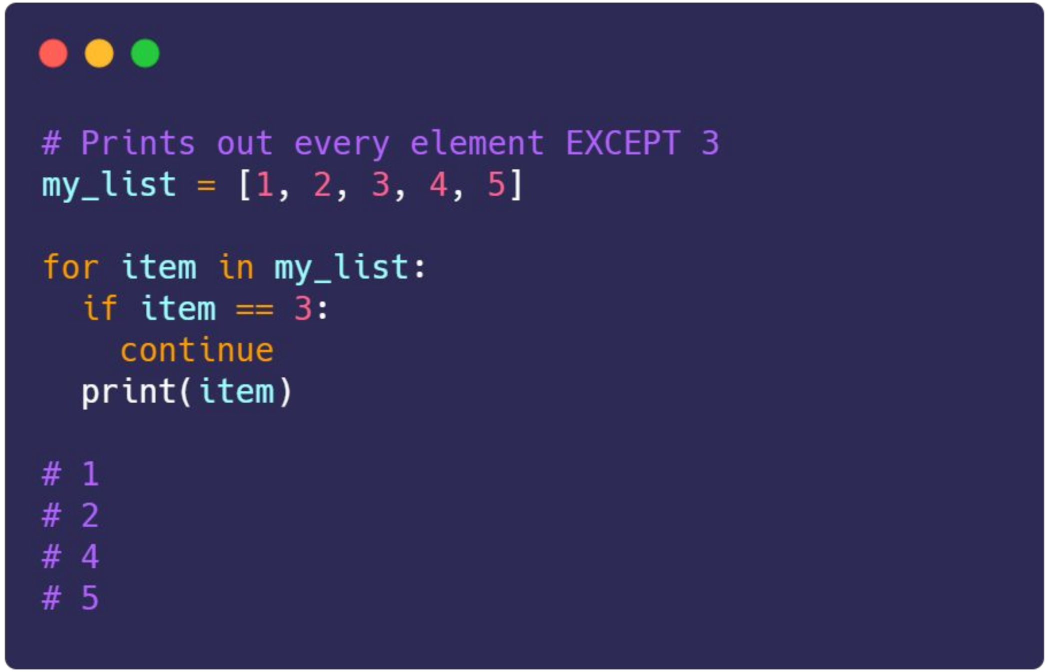
- **break** : ends a loop early



```
# Prints out each element until it gets to 3
my_list = [1, 2, 3, 4, 5]

for item in my_list:
  if item == 3:
    break
  print(item)

# 1
# 2
```

continue

- continue: ignores a specific element and moves on to next



```
# Prints out every element EXCEPT 3
my_list = [1, 2, 3, 4, 5]

for item in my_list:
  if item == 3:
    continue
  print(item)

# 1
# 2
# 4
# 5
```

Now Try This

1. Create a list called **`alphabet`** and fill with all letters of the English alphabet (i.e. **`[‘a’, ‘b’, ‘c’, … ]`**). Use either a **for loop** or a **while loop** to complete the following:
   a. Print out every 4th letter starting from **`‘a’.`** In other words, the first letter printed should be **`‘a’,`** followed by **`‘d’.`**
   b. Skip the letter **`‘l’.`** (**`‘L’`**)
   c. Exit the loop after the 16th letter has been printed
   d. **HINT:** Use step-size to traverse the list. Example: **`my_list[0::2]`** prints every other element. Use break or continue accordingly to exit the loop or skip a step in the loop.
2. s

Bit Project

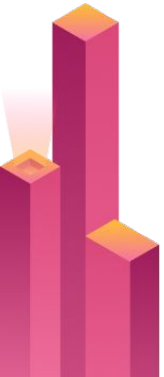
Now Try This

2. You are copying files from your USB to your laptop. Let the lists **`usb`** and **`laptop`** represent the two devices. Fill **`usb`** with at least 20 elements of any data type. Then, use either a **for loop** or a **while loop** to complete the following:

   a. Remove an item from **`usb`**
   b. Append that item to **`laptop`**
   c. Continue (a) and (b) until the **`usb`** list is empty and the **`laptop`** list has all of the elements.
   d. **HINT:** Use the **`.pop()`** and **`.append()`** methods to remove and add items to a list, and use **`len()`** to determine the number of elements.

Bit Project

Functions



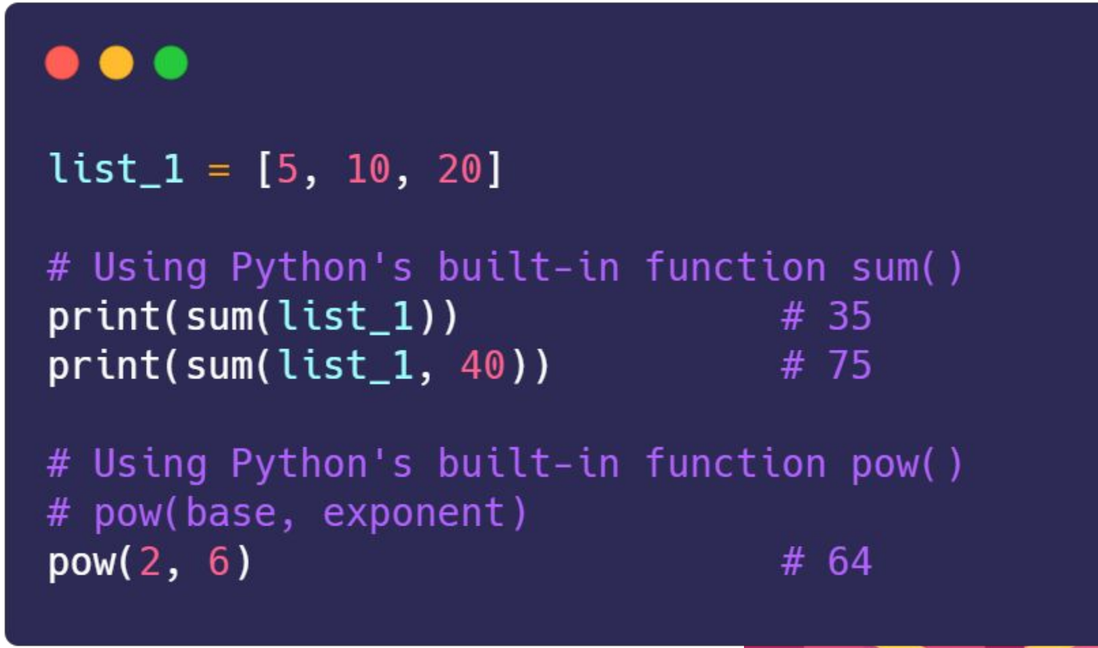
Bit Project

Intro to Functions

- **Functions** allow us to write a chunk of code once, and run multiple times.
  - Format:

`function_name (arg1, arg2)`

- **Parameters:** placeholders for variables within a function



```
list_1 = [5, 10, 20]

# Using Python's built-in function sum()
print(sum(list_1))                      # 35
print(sum(list_1, 40))                  # 75

# Using Python's built-in function pow()
# pow(base, exponent)
pow(2, 6)                               # 64
```

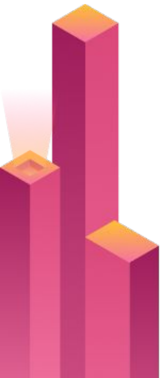
Bit Project

Now Try This

What do you think happens when you don’t pass in the right amount of parameters for a function?

Bit Project

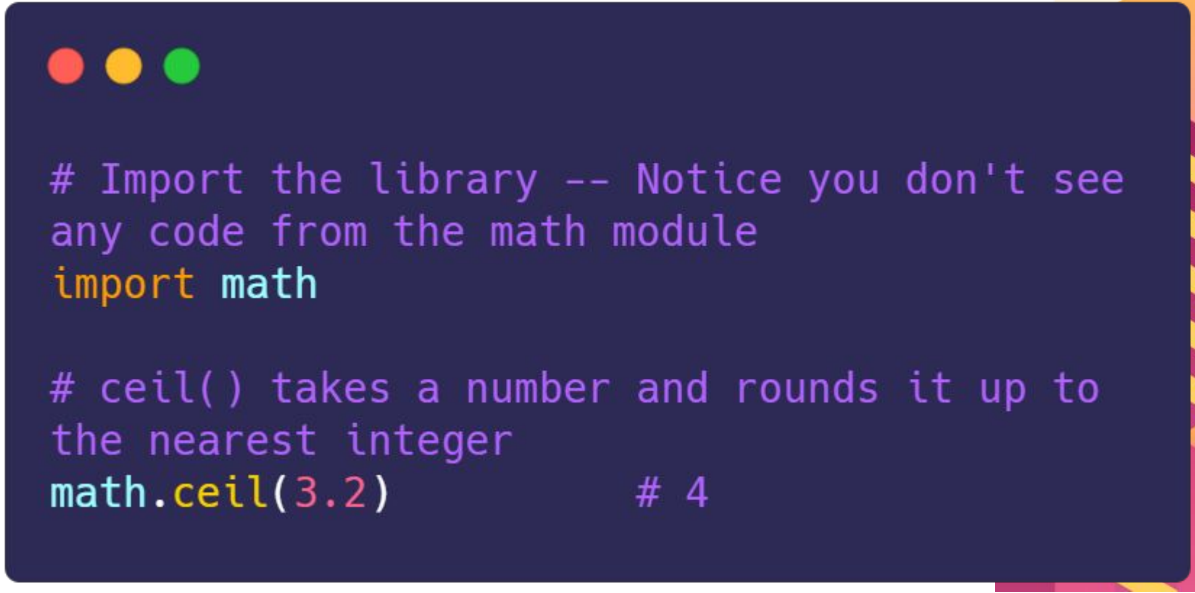
Modules and Packages



Bit Project

Understanding Modules

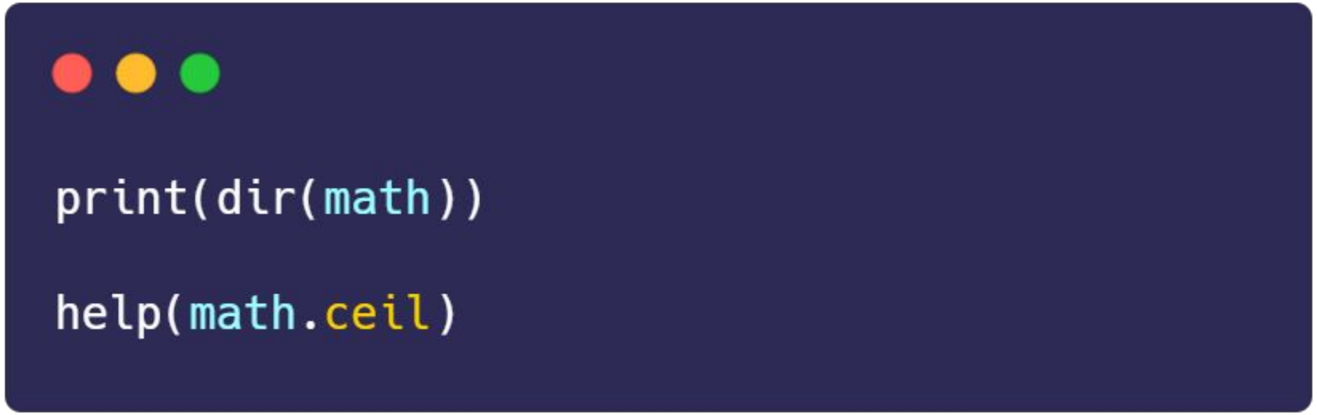
- **Modules**: full Python programs
    - Allow further code reusability
    - Prewritten functions can be `import`ed for use
- Examples:
    - `math` module



```python
# Import the library -- Notice you don't see
any code from the math module
import math

# ceil() takes a number and rounds it up to
the nearest integer
math.ceil(3.2)          # 4
```

Exploring Built-In Modules

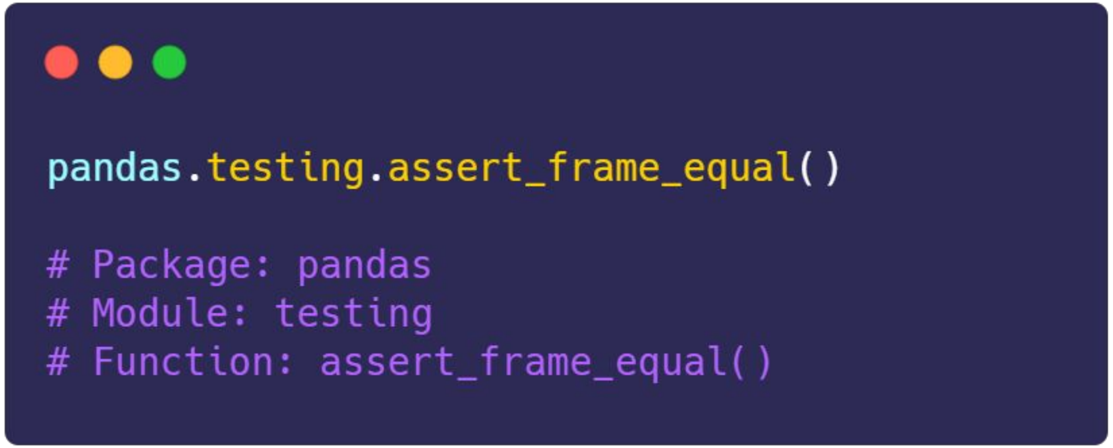
- **dir():** tells you **what functions** are in a package
- **help():** provides **brief description** about **what the function does** and the **parameters** required



```
print(dir(math))

help(math.ceil)
```

Understanding Packages

- **Packages:** folders to store **modules** or to store **other packages**
  - Allows further code reusability to **organize multiple modules** with a **common theme**



```
pandas.testing.assert_frame_equal()

# Package: pandas
# Module: testing
# Function: assert_frame_equal()
```

Bit Project

Takeaways

- Solid foundation for:
    - General programming with Python
    - Using Python's data science tools and libraries
- Next week:
    - Pandas (Python data library to organize and manipulate data)

Bit Project

Follow Us!

@bitprojectorg

@bitprj

@BitProject

@BitProject

join.bitproject.org