Intro to Intermediate Python

Learning Objectives

After this lesson, you will be able to:

- Confidently recap the previous units.
- Describe key components of the upcoming unit.

Leveling Up

You’re leveling up!

You have the proper foundation. Now, let’s check how you’re doing.

Let's Review: Lists

- A collection of items stored in a single variable.
- Created with square brackets (`[]`).
- Begin counting at `0`.

```
my_queens = ["Cersei", "Daenerys", "Arwen", "Elsa", "Guinevere"]

step_counts_this_week = [8744, 5256, 7453, 3097, 4122, 2908, 6720]


# We can also mix types.

weird_list = [1, "weird", ["nested list"], "eh?"]
```

Challenge: *Can you recall how to slice a section of the list? For example, items 2 through 5 of* `step_counts_this_week`*?*

Answer: Lists Challenge

- Python uses a `:` to represent a range of indices.
- Beware of off-by-one errors!

```
step_counts_this_week = [8744, 5256, 7453, 3097, 4122, 2908, 6720]

days_2_thru_5 = step_counts_this_week[2:6] # Items 2, 3, 4, and 5
```

Pro tip: *It's `6` instead of `5` because the range is exclusive.*

Let's Review: Loops and Iteration

What about looping a list?

```
my_queens = ["Cersei", "Daenerys", "Arwen", "Elsa", "Guinevere"]


for queen in my_queens:
    print(queen, "is the most powerful queen!")
```

Challenge: *What if I want to loop from 1 to 10 and print out the numbers? How do I do this without a data structure to loop over?*

Answer: Loops Challenge

To loop 1–10 without a data structure:

```
# Remember, "i" is a common name for a counter/index in programming!
for i in range(1, 11):
    print(i)
```

- Why do you think we put `11` in the code?
- What values does this print?

Let's Review: Sets

- Lists that don't have duplicates.
- Created with curly braces (`{}`) or from lists with the `set()` function.
- Aren't indexed — elements are in any order!
- Handy for storing emails, user names, and other unique elements.

```
email_set = {'my_email@gmail.com', 'second_email@yahoo.com', "third_email@hc

# Or from a list:

my_list = ["red",  "yellow", "green", "red", "green"]

my_set = set(my_list)

# => {"red",  "yellow", "green"}
```

Let’s Review: Tuples

- Lists that can’t be changed!
- Created with parentheses (`()`).
- Can’t add, pop, remove, or otherwise change elements after creation.

```
rainbow_colors_tuple = ("red", "orange", "yellow", "green", "blue", "indigo"
```

Let's Review: Dictionaries

- A collection of key-value pairs.
- Created with curly braces (`{key: value, key: value}`).
- Values can be anything!

```
my_puppy = {
    "name": "Fido",
    "breed": "Corgi",
    "age": 3,
    "vaccinated": True,
    "fave toy": ["chew sticks", "big sticks", "any sticks"]
}
```

Challenge: *Can you recall how to iterate (loop) over each key of* `my_puppy` *and print out both the key and the corresponding value?*

Answer: Dictionaries Challenge

Iterating a dictionary is similar to a list:

```
for key in my_puppy:
    print(key, "-", my_puppy[key])
```

Outputs:

```
name - Fido
breed - Corgi
age - 3
vaccinated - True
fave toy - chew sticks
```

Let's Review: Functions

- Bits of code that can be used repeatedly.
- Enable DRY — Don't Repeat Yourself.
- Declared with `def`, `()`, and `:`.
- Declare the function *above* the function call!

```
# Function definition:
def say_hello():
    print("hello!")


# Run the function three times.
say_hello()
say_hello()
say_hello()
```

run ▶

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
```

run ▶

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
```

Let's Review: Return Statements

- Bring data out of a function.
- Cause the function to exit.
- Aren't a `print` statement!

```
def multiply(x, y):
    return x * y


result = multiply(3, 4) # Result is now equal to 12.
```

Let's Review: Classes

- Templates (aka, blueprints) for objects.
- Can contain methods and/or variables.
- `self` is a reference to the created object.

```
class Animal():

    def __init__(self):

        self.energy = 50

def get_status(self):

    if self.energy < 20:

        print("I'm hungry!")

    elif self.energy > 100:

        print("I'm stuffed!")

    else:

        print("I'm doing well!")
```

Challenge: *How do you declare a new* `Animal`*?*

Answer: Classes

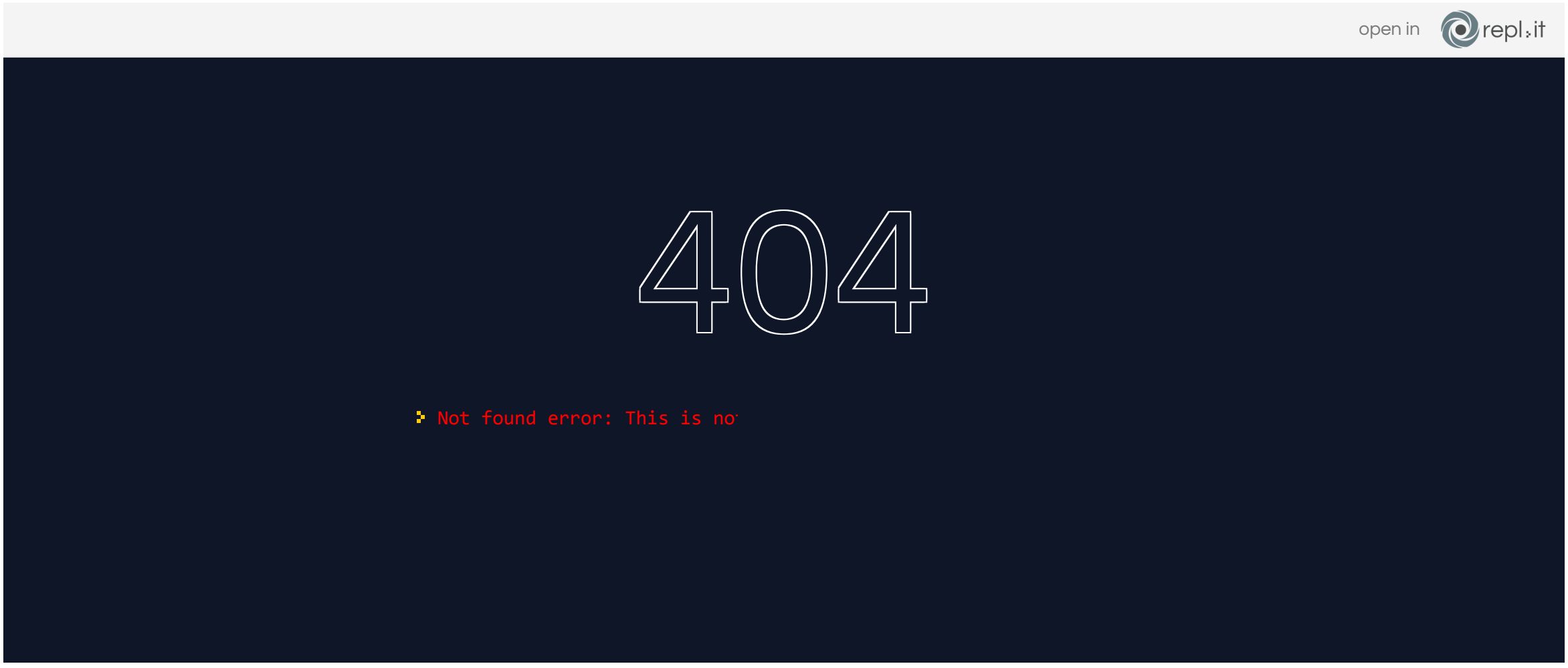
Declaring a new `Animal` from the class:

```
my_animal = Animal() # Creates a new Animal instance.
my_animal.get_status() # Prints "I'm doing well!"
my_animal.energy += 100 # We can access properties!
my_animal.get_status() # Prints "I'm stuffed!"
```

run ▶

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
```

Inheritance: Answer



Knowledge Check

We’re about to move on to the next unit: Intermediate Python.

Any questions?

> *Don’t be shy! If you have a question, so do others!*

Switching Gears: Preview

The next unit covers many topics, including:

- User input
- File I/O
- Abstraction
- Modules and libraries
- APIs

You don’t need to memorize them now! This is just an overview.

User Input and File I/O

You've seen this a few times already with `input()`.

We'll build real interactions between your Python programs and other files — or the person using your app!

Abstraction

Python has built-in functions for performing common tasks.

You've seen things like `my_list.len()`, which tells you the length of a list.

But Python has more specialized built-in functions, like chaining lists together:

```
food = ['pizza', 'tacos', 'sushi']

colors = ['red', 'green']

# => lists_chained =['pizza', 'tacos', 'sushi', 'red', 'green']
```

This helps you get complex things done more quickly.

We'll learn several of these.

run ▶

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
```

run ▶

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
```

Summary and Q&A

We reviewed topics from earlier lessons:

- Lists, sets, tuples, and dictionaries.
- Loops and iteration.
- Functions, parameters, and return statements.
- Classes and inheritance.

We brushed the surface on some upcoming topics:

- User input and file I/O.
- Abstraction.
- Modules and libraries.
- APIs.

Let’s jump in to it!

Additional Reading and Resources

Now that you have an understanding of basic programming, here are some cool people to read about:

- **Ada Lovelace:** Regarded as the first programmer.
- **Alan Turing:** Considered the father of theoretical computer and artificial intelligence; helped crack the enigma code during World War II.
- **Linus Torvalds:** Creator of Linux OS and Git.