

The background of the entire page is a repeating pattern of stylized, teal-colored leaves or feathers. These elements are scattered across the white background, with some appearing more prominently than others. The leaves have a fine, linear texture, suggesting a sketch or a digital pattern.

Raspberry Pi for Beginners

LESSON 8

MAKERHOUSE
EMPOWERING MAKERS

TABLE OF CONTENTS

1	<i>Introduction</i>	2
1.1	Formatting Numbers	2
1.2	Formatting Dates	2
1.3	Returning More Than One values	3
1.4	Defining a Class	3
1.5	Defining a Method	4
1.6	Inheritance	4
1.7	Writing to a File	5
1.8	Reading from File	5
1.9	Pickling	5
1.10	Handling Exceptions	6
1.11	Using Module	7
1.12	Random Numbers	8
2	<i>Challenge: A Python Game of Cat and Mouse</i>	9
2.1	The Code	9

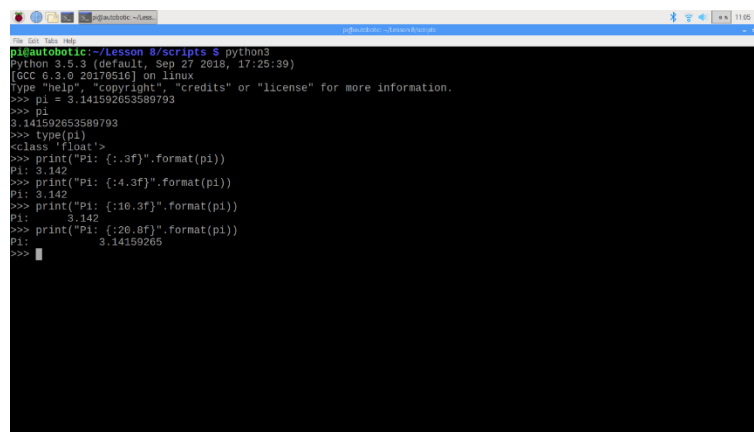
GOING DEEPER INTO PYTHON

1 INTRODUCTION

Starting from Lesson 5 until 7, we have learned a lot about Python. To add, we have been created a various text-based game using Python. It's such a wonderful feeling, right? We not only grabbed the gist of Python language but also implemented it in fun way through games. Congratulate yourself my dear friends.

Now you have the basics of Python yet become more advanced by going deeper into Python with this lesson; Lesson 8. Be prepared for explore more advanced concepts in the Python language—in particular, object-oriented Python, reading and writing files, handling, exceptions, using modules, and Internet programming

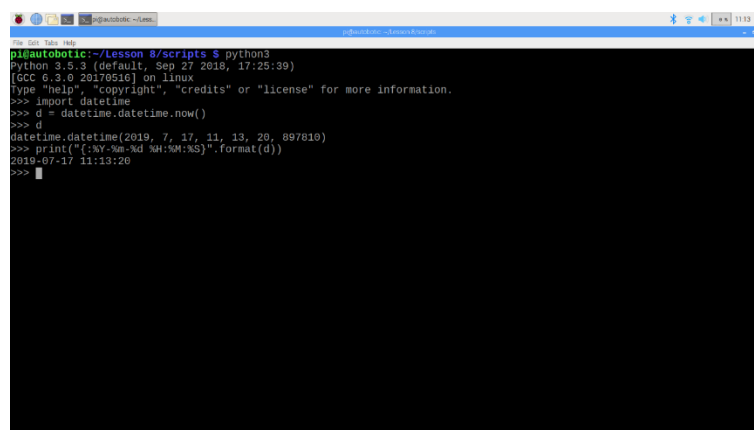
1.1 FORMATTING NUMBERS



```
File Edit View Help
Python 3.5.3 Shell
p1@autobotic:~/Lesson 8/scripts$ python3
Python 3.5.3 (default, Sep 27 2016, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> pi = 3.141592653589793
>>> pi
3.141592653589793
>>> type(pi)
<class 'float'>
>>> print("Pi: {:.3f}".format(pi))
Pi: 3.142
>>> print("Pi: {:.4f}".format(pi))
Pi: 3.142
>>> print("Pi: {:.10.3f}".format(pi))
Pi: 3.142
>>> print("Pi: {:.20.8f}".format(pi))
Pi: 3.14159265
>>>
```

Figure 1: Formatting numbers to a certain number of decimal places.

1.2 FORMATTING DATES



```
File Edit View Help
Python 3.5.3 Shell
p1@autobotic:~/Lesson 8/scripts$ python3
Python 3.5.3 (default, Sep 27 2016, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> import datetime
>>> d = datetime.datetime.now()
>>> d
datetime.datetime(2019, 7, 17, 11, 13, 20, 897810)
>>> print("[%Y-%m-%d %H:%M:%S]".format(d))
2019-07-17 11:13:20
>>>
```

Figure 2: Converting a date into a string and format it in a certain way.

1.3 RETURNING MORE THAN ONE VALUES

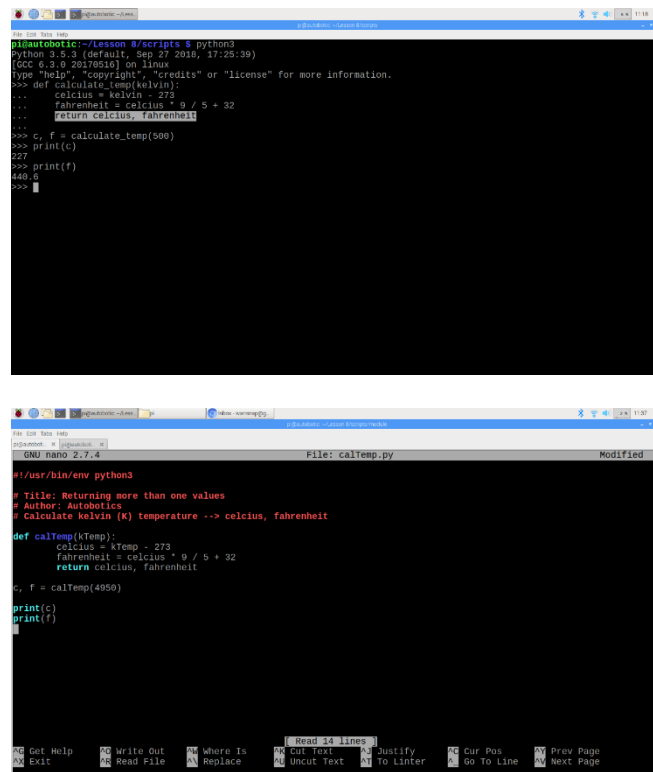


Figure 3: Writing a function that returns more than one value

1.4 DEFINING A CLASS

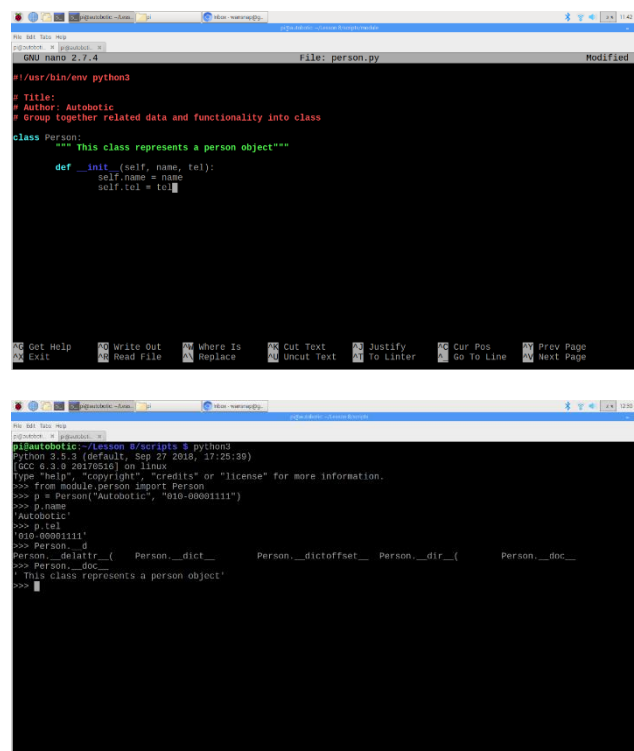
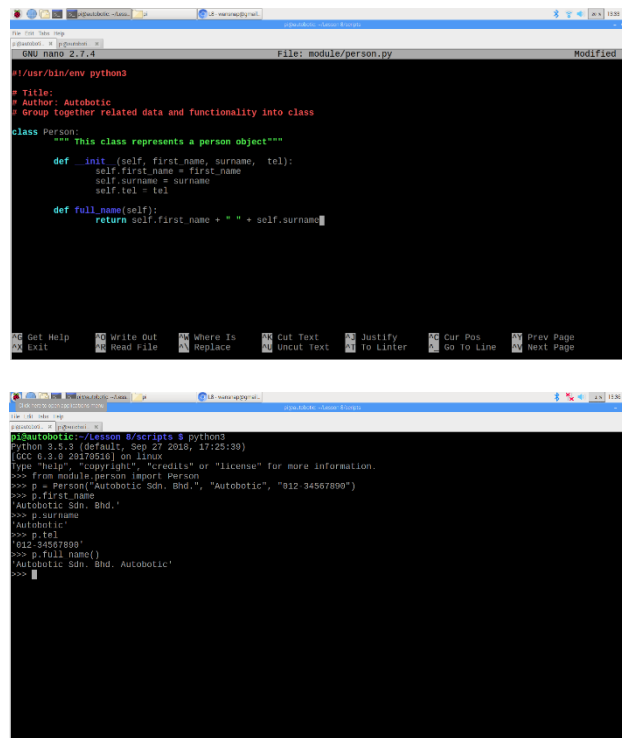


Figure 4: Grouping together related data and functionality into a class.

1.5 DEFINING A METHOD



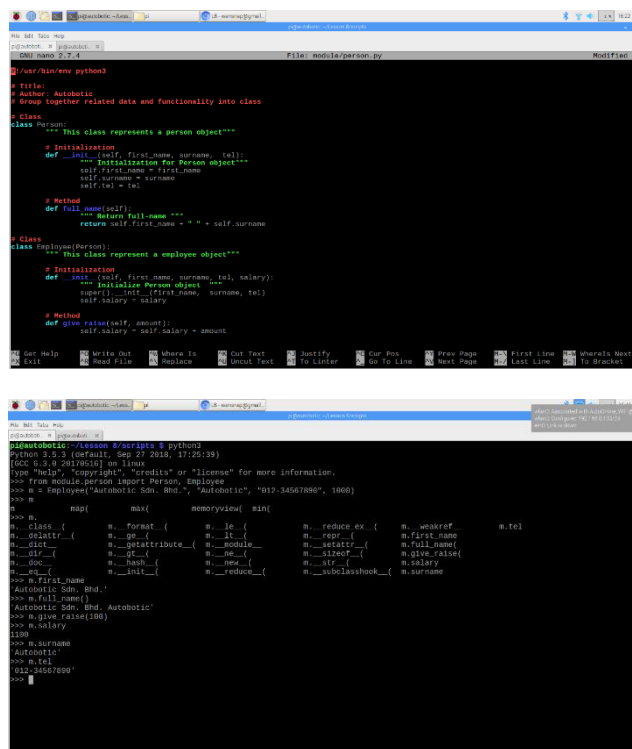
```
#!/usr/bin/env python3
# Title:
# Author: Autobotic
# Group together related data and functionality into class
# Class
class Person:
    """ This class represents a person object """
    def __init__(self, first_name, surname, tel):
        self.first_name = first_name
        self.surname = surname
        self.tel = tel
    def full_name(self):
        return self.first_name + " " + self.surname

# Test
p = Person("Autobotic Sdn. Bhd.", "Autobotic", "012-34567890")
p.first_name
p.surname
p.tel
p.full_name()
```

```
Autobotic:~/Lesson 8/scripts $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> from module.person import Person
>>> p = Person("Autobotic Sdn. Bhd.", "Autobotic", "012-34567890")
>>> p.first_name
'Autobotic Sdn. Bhd.'
>>> p.surname
'Autobotic'
>>> p.tel
'012-34567890'
>>> p.full_name()
'Autobotic Sdn. Bhd. Autobotic'
>>>
```

Figure 5: Adding a method to a class.

1.6 INHERITANCE



```
#!/usr/bin/env python3
# Title:
# Author: Autobotic
# Group together related data and functionality into class
# Class
class Person:
    """ This class represents a person object """
    def __init__(self, first_name, surname, tel):
        self.first_name = first_name
        self.surname = surname
        self.tel = tel
    def full_name(self):
        return self.first_name + " " + self.surname

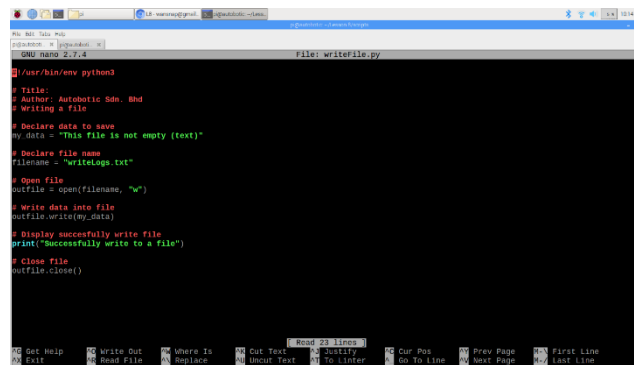
# Class
class Employee(Person):
    """ This class represent a employee object """
    def __init__(self, first_name, surname, tel, salary):
        """ Initialize Person object """
        super().__init__(first_name, surname, tel)
        self.salary = salary
    def give_raise(self, amount):
        self.salary = self.salary + amount

# Test
m = Employee("Autobotic Sdn. Bhd.", "Autobotic", "012-34567890", 1000)
m
m.first_name
m.surname
m.tel
m.full_name()
m.give_raise(100)
m.salary
m.give_raise(100)
m.salary
```

```
Autobotic:~/Lesson 8/scripts $ python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> from module.person import Person, Employee
>>> m = Employee("Autobotic Sdn. Bhd.", "Autobotic", "012-34567890", 1000)
>>> m
<module.person.Employee object at 0x7f8b1c1c1c1c>
>>> m.first_name
'Autobotic Sdn. Bhd.'
>>> m.surname
'Autobotic'
>>> m.tel
'012-34567890'
>>> m.full_name()
'Autobotic Sdn. Bhd. Autobotic'
>>> m.give_raise(100)
>>> m.salary
1100
>>> m.give_raise(100)
>>> m.salary
1200
>>>
```

Figure 6: Specialized version of an existing class.

1.7 WRITING TO A FILE



```
File: writefile.py
#!/usr/bin/env python3
# Title:
# Author: Autobot Sdn. Bhd
# Writing a file

# Declare data to save
my_data = "This file is not empty (text)"

# Declare file name
filename = "logfile.txt"

# Open file
outfile = open(filename, "w")

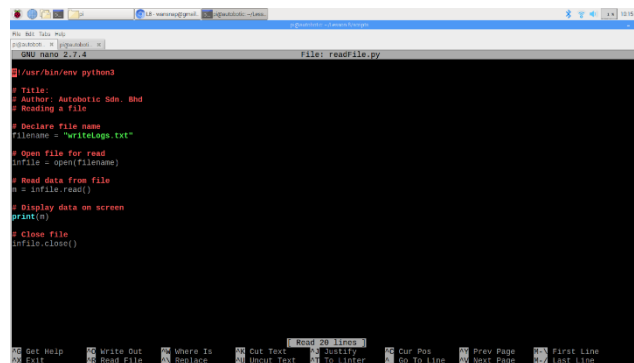
# Write data into file
outfile.write(my_data)

# Display successfully write file
print("Successfully write to a file")

# Close file
outfile.close()
```

Figure 7: Writing something to a file.

1.8 READING FROM FILE



```
File: readfile.py
#!/usr/bin/env python3
# Title:
# Author: Autobot Sdn. Bhd
# Reading a file

# Declare file name
filename = "logfile.txt"

# Open file for read
infile = open(filename)

# Read data from file
a = infile.read()

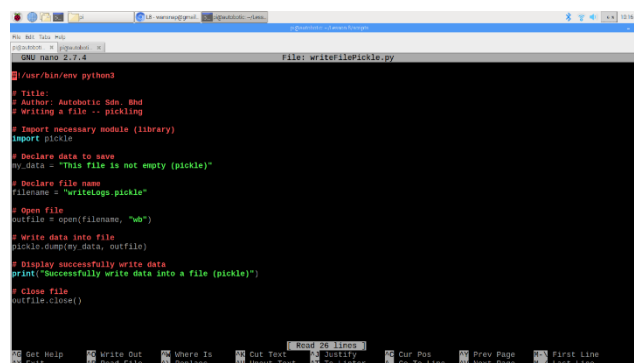
# Display data on screen
print(a)

# Close file
infile.close()
```

Figure 8: Reading the contents of a file into a string variable.

1.9 PICKLING

Another saving and reading entire contents of a data structure to/from a file -useful for the next time the program is run.



```
File: writefilePickle.py
#!/usr/bin/env python3
# Title:
# Author: Autobot Sdn. Bhd
# Writing a file -- pickling
import pickle

# Declare data to save
my_data = "This file is not empty (pickle)"

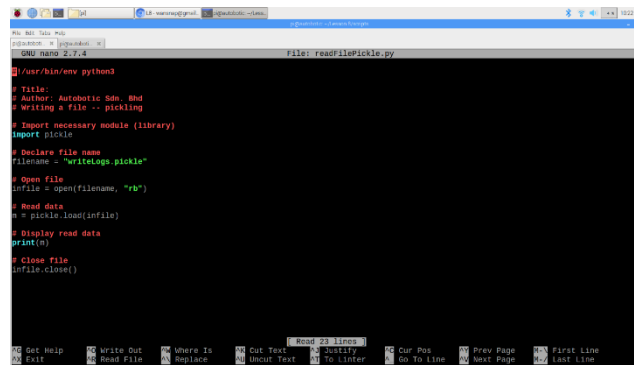
# Declare file name
filename = "logfile.pickle"

# Open file
outfile = open(filename, "wb")

# Write data into file
pickle.dump(my_data, outfile)

# Display successfully write data
print("Successfully write data into a file (pickle)")

# Close file
outfile.close()
```



```

#!/usr/bin/env python3
# Title:
# Author: Autobot Sdn. Bhd
# Writing a file -- pickling
# Import necessary module (library)
import pickle

# Declare file name
filename = "writeLogs.pickle"

# Open file
infile = open(filename, "rb")

# Read data
s = pickle.load(infile)

# Display read data
print(s)

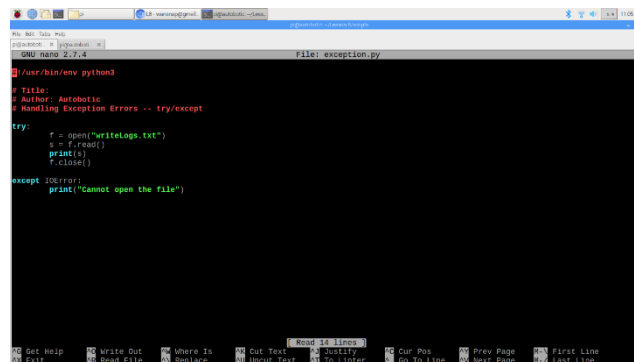
# Close file
infile.close()

```

Figure 9: Reading and writing the contents of a file using pickle module.

1.10 HANDLING EXCEPTIONS

To catch the error or exception and display a more user-friendly error message -- use Python's **try/except** construct or may to have **else** and **finally** clauses in the error handling.

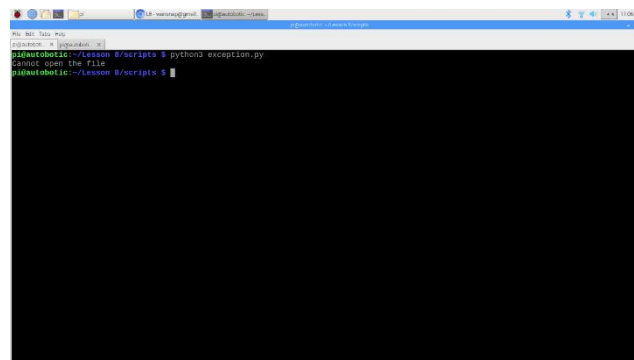


```

#!/usr/bin/env python3
# Title:
# Author: Autobot Sdn. Bhd
# Handling Exception Errors -- try/except

try:
    f = open("writeLogs.txt")
    s = f.read()
    print(s)
    f.close()
except IOError:
    print("Cannot open the file")

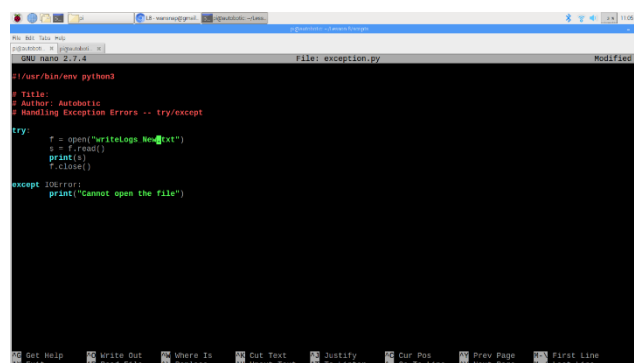
```



```

pi@autobotic:~/Lesson 8/scripts $ python3 exception.py
Cannot open the file
pi@autobotic:~/Lesson 8/scripts $

```



```

#!/usr/bin/env python3
# Title:
# Author: Autobot Sdn. Bhd
# Handling Exception Errors -- try/except

try:
    f = open("writeLogs New.txt")
    s = f.read()
    print(s)
    f.close()
except IOError:
    print("Cannot open the file")

```

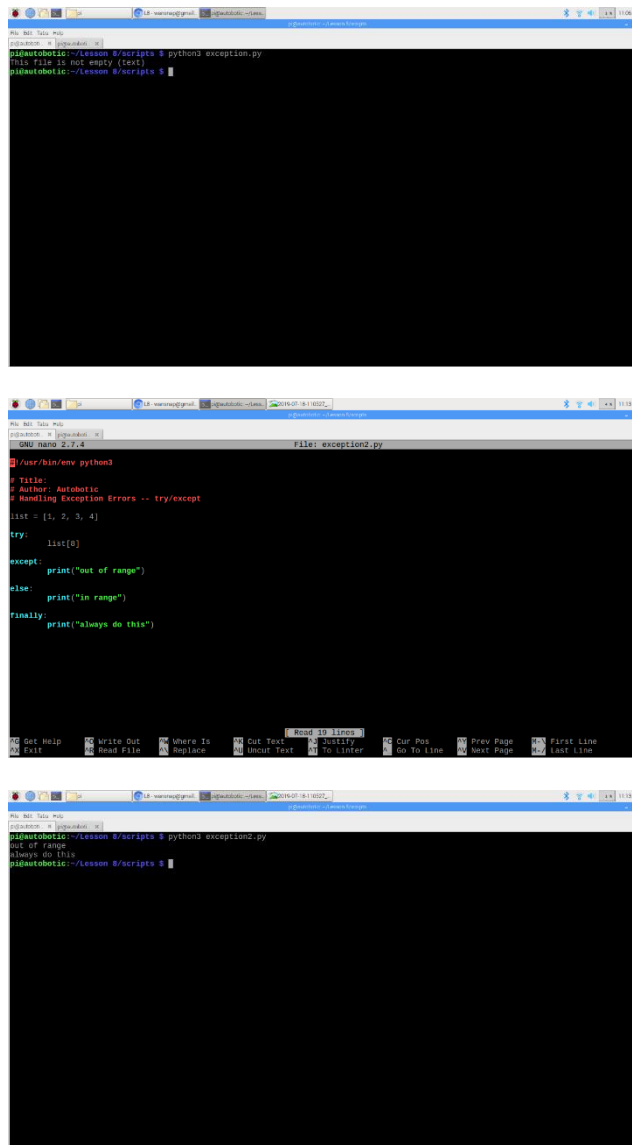
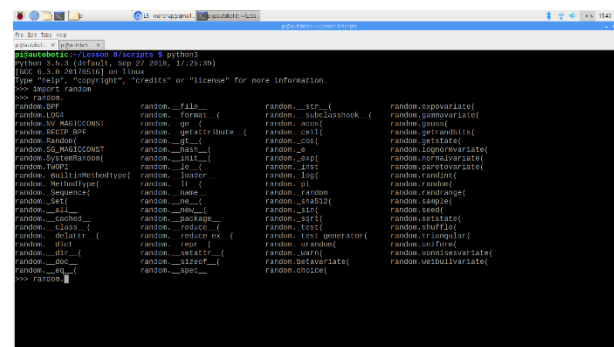


Figure 10: Reading and writing the contents of a file using pickle module.

1.11 USING MODULE

We eventually used various modules (libraries) – turtles, random – by using the **import** command.



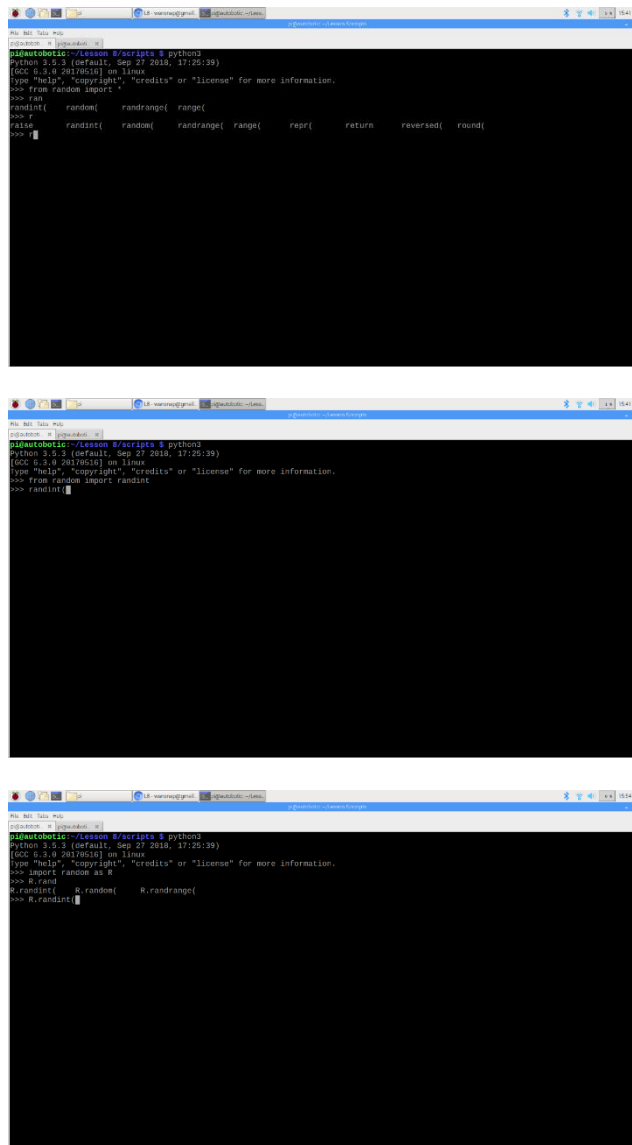


Figure 11: Various way to import (include) modules (libraries).

1.12 RANDOM NUMBERS

Generating a random thing such as number between a range of numbers by importing the random modules (libraries).

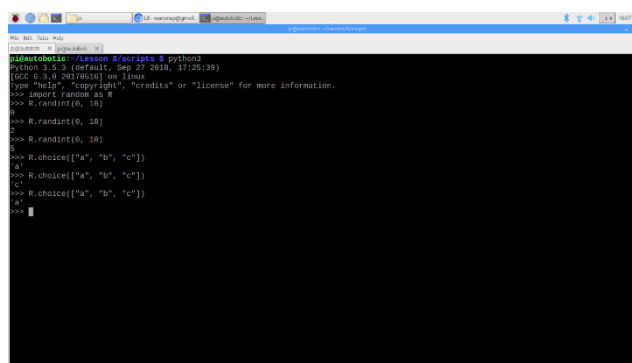
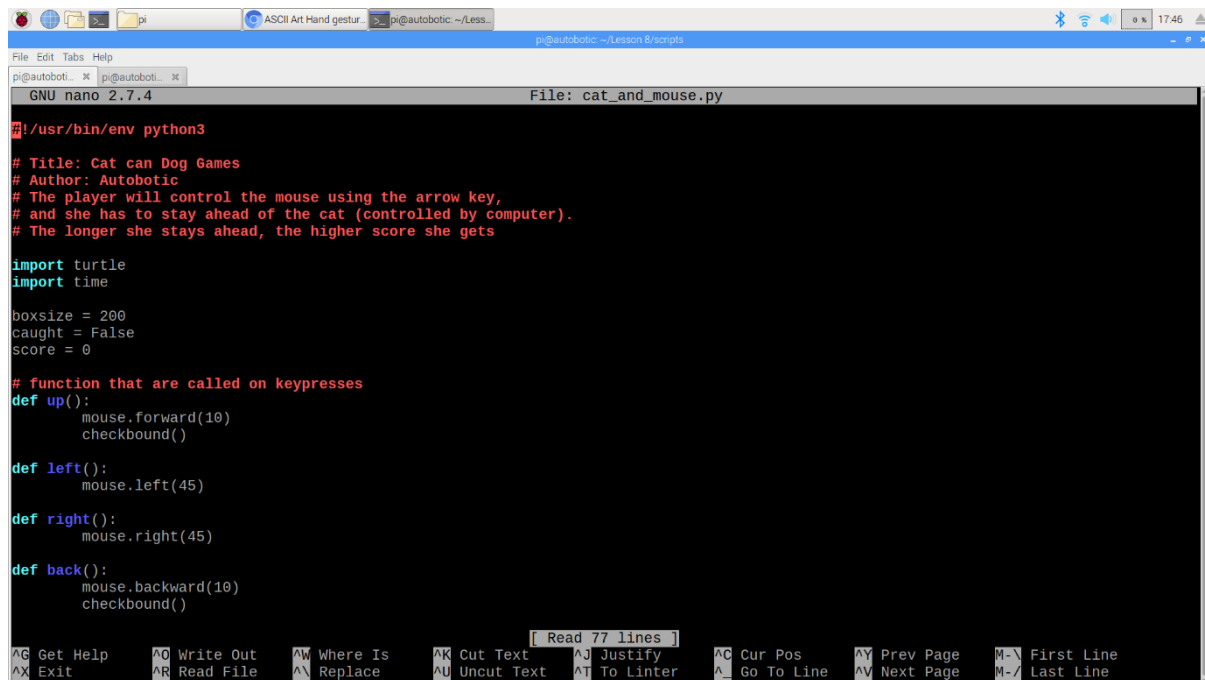


Figure 12: Importing the random module for random numbers or choice of selections.

2 CHALLENGE: A PYTHON GAME OF CAT AND MOUSE

This time we are going to make a game of cat and mouse. The player will control the mouse using the arrow keys and must stay ahead of the cat (controlled by the computer). The longer the mouse stays ahead, the higher the score it gets.

2.1 THE GAMES



```

GNU nano 2.7.4 File: cat_and_mouse.py

#!/usr/bin/env python3

# Title: Cat can Dog Games
# Author: Autobotic
# The player will control the mouse using the arrow key,
# and she has to stay ahead of the cat (controlled by computer).
# The longer she stays ahead, the higher score she gets

import turtle
import time

boxsize = 200
caught = False
score = 0

# function that are called on keypresses
def up():
    mouse.forward(10)
    checkbound()

def left():
    mouse.left(45)

def right():
    mouse.right(45)

def back():
    mouse.backward(10)
    checkbound()
  
```

Figure 13: Scripting cat and mouse games in Python

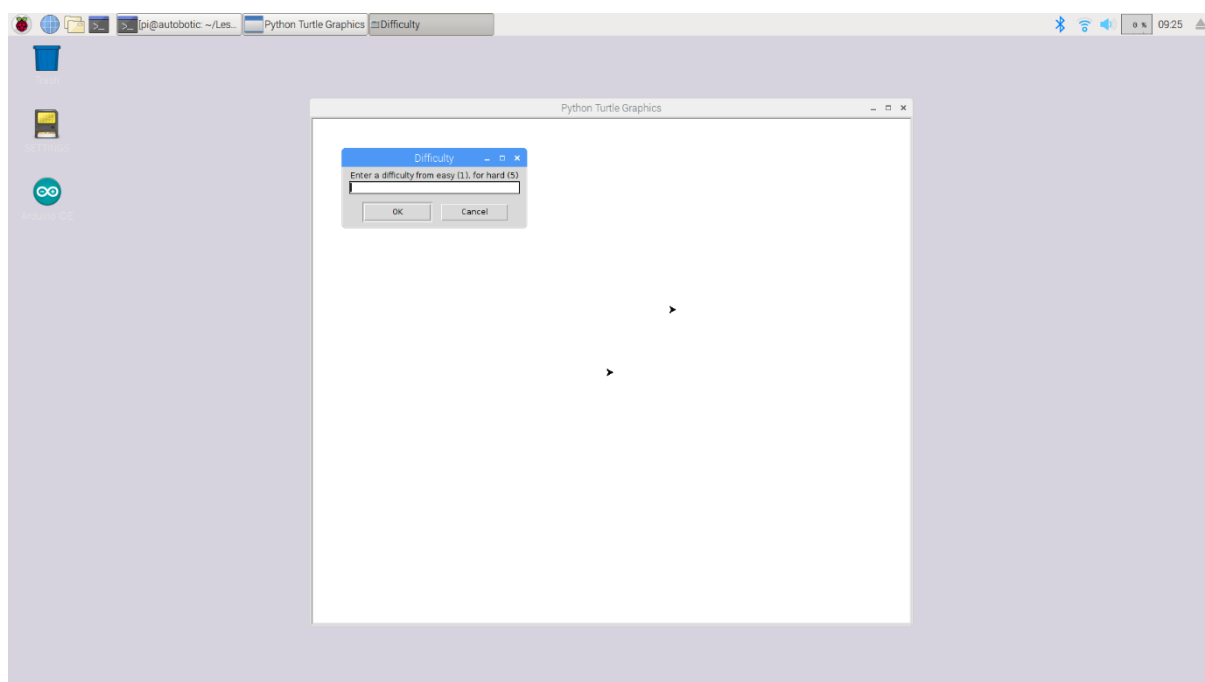


Figure 13: Cat and Dog Game Interface

2.1.1 THE FULL CODES

```
#!/usr/bin/env python3

# Title: Cat can Dog Games
# Author: AutobotiC
# The player will control the mouse using the arrow key,
# and she has to stay ahead of the cat (controlled by computer).
# The longer she stays ahead, the higher score she gets

# importing important modules (libraries) – Turtle and time
import turtle
import time

# declare variable
boxsize = 200
caught = False
score = 0

# function that are called on keypresses
def up():
    mouse.forward(10)
    checkbound()

def left():
    mouse.left(45)

def right():
    mouse.right(45)

def back():
    mouse.backward(10)
    checkbound()

def quitTurtles():
    windows.bye()
```

```
# stop the mouse from leaving the square set by box size
def checkbound():
    global boxsize
    if mouse.xcor() > boxsize:
        mouse.goto(boxsize, mouse.ycor())
    elif mouse.xcor() < -boxsize:
        mouse.goto(-boxsize, mouse.ycor())
    elif mouse.ycor() > boxsize:
        mouse.goto(mouse.xcor(), boxsize)
    elif mouse.ycor() < -boxsize:
        mouse.goto(mouse.xcor(), -boxsize)

# set up screen
window = turtle.Screen()
mouse = turtle.Turtle()
cat = turtle.Turtle()
mouse.penup()
mouse.penup()
mouse.goto(100, 100)

# add key listeners
window.onkeypress(up, "Up")
window.onkeypress(left, "Left")
window.onkeypress(right, "Right")
window.onkeypress(back, "Down")
window.onkeypress(quitTurtles, "Escape")

difficulty = window.numinput("Difficulty", "Enter a difficulty from easy (1), for hard (5)", minval=1, maxval=5)

window.listen()

# main loop
# note how to changes with difficulty
while not caught:
```

```
cat.setheading(cat.towards(mouse))

cat.forward(8 + difficulty)

score = score + 1

if cat.distance(mouse) < 5:
    caught = True
    time.sleep(0.2 - (0.01 * difficulty))

window.textinput("GAME OVER", "Well done. You scored:" + str(score * difficulty))

window.bye()
```

Listing 1: Full code of cat and mouse games in Python