python-5

September 15, 2020

0.1 # Introduction to Programming – Python

0.2 Course 5 – Files, Modules and Exceptions

0.2.1 ESSEC Business School

gael.guibon@gmail.com

gael.guibon@telecom-paris.fr

1 Summary

- **instructions** : `print()`
- **basic operators** : logic (`and or`), math (`+ = - /`), membership (`in not in`)
- **variables** : `name = city city = 'Cergy'`
- **types concepts and basics** and conversion : `type() str() bool() int() float()`
- **conditions and nested conditions** : indentations !
- **lists and lists functions** : `list()`, `myList = ['hello', 'world']`, `myList.append()`, `len(myList)` etc.
- **loops** : while `while conditionIsTrue:` and for `for element in elementList:`, `break`, `continue`
- **dictionaries** : `dict()`, `{key:value}`, nested dict `{key: {key:value} }`, `list(d.keys())`, `for k,v in d.items()`
- **functions** : `def myFunction(arg1): return True`, variable scope, docstrings `""" my doc """`, etc.
- Mini textual game

2 Install local python environnement

At home you may want to use Python locally. Here are the steps: 1. Download the latest Python (Python3 not 2) from here: https://www.python.org/downloads/ 2. Install it (if you are using Ubuntu 16.04 you already have Python installed)

Start using it: - Open terminal(unix/mac) or CommandLine(windows), type `python` to start an interactive python environnement - Create a file named `my_super_program.py` and type `print('hello')` inside. Execute this file by typing `python3 my_super_program.py`.

To code you may need an IDE for smoother coding. I would suggest Visual Studio Code. For a python only IDE the best one would be PyCharm.

3 File handling

A program often need to access a file from your file system. For this you need: 1. A **file path** (*i.e.* the location). It can be: - absolute: `/home/gael/Documents/my_super_file.txt` - relative: `Documents/my_super_file.txt` (when you want to access it from `/home/gael/` 2. A **function** `open()` to handle files: - **Create** a file from scratch: `myFile = open(file_path, 'w')` - **Write** in a file opened with 'w': `myFile.write('hello world')` - **Read** a file: `myFile = open(file_path, 'r')` - **Close** a file: `myFile.close()`

Create a file named 'test.txt':

[ ]:
```
myFile = open('test.txt', 'w') # create the file with 'w' access (w = write) and␣
↪get the file object variable
```

Write a String content into the file:

[ ]:
```
myFile.write('All hail Python') # write function is available if the file was␣
↪opened with 'w'
```

Always close a file once you are done with it:

[ ]:
```
myFile.close()
```

Read a file and get the value into a variable:

[ ]:
```
myFile = open('test.txt', 'r') # open an existing file with 'r' (r = read)
fileContent = myFile.read() # put the content into a variable
myFile.close()
print(fileContent, type(fileContent)) # display the content
```

Append content to a file:

[ ]:
```
myFile = open('test.txt', 'a') # a = append
myFile.write('\nAnd Java !')
myFile.close()
```

Read the file again to display its new content

[ ]:
```
myFile = open('test.txt', 'r')
newContent = myFile.read()
myFile.close()
print(newContent)
```

Shortcut to autoclosing files:

[ ]:
```
with open('test.txt', 'a') as myFile:
    myFile.write('hey it works !')
# leaving the with indentation auto-close the file
with open('test.txt', 'r') as myFile:
    print( myFile.read() )
```

4 Modules

Python possess a huge ecosystem with modules for everything. A lot of them are already available, such as the `json` module. The `json` allows you to insert a dictionary or a list of dictionaries into a file, and to access it later on.

To use modules you need: 1. to import them `import json` 2. to look at their documentation: https://docs.python.org/fr/3/library/json.html

Create a dict and put it inside a json file:

[ ]:
```
import json

events = {'run': {'message': 'You tried to run but the dog bites you.'}, 'feed':␣
↪{'message':'You fed the dog with your arm. That hurt!'} }

with open("test.json", "w") as myFile:
    json.dump(events, myFile)
```

Now let us read the json file in a normal way:

[ ]:
```
myFile = open('test.json', 'r')
content = myFile.read()
print(type(content))
print(content)
```

The content is a String but we wanted to parsed it as a dictionary. To do so we can use the `json` module to open it:

[ ]:
```
myFile = open('test.json', 'r')
content = json.load(myFile)
print(type(content))
#print(content)
print(content['run'])
```

The `json` module can read or write from String representations:

[ ]:
```
contentString = '{"type":"I am a string dict, not a real one"}'
content = json.loads( contentString )
myFile.close()
```

```
print('from', type(contentString), 'to', type(content))

eventsString = json.dumps( events )
print('from', type(events) , 'to', type(eventsString) )
```

5 Exceptions

Sometimes errors happen, and you want to handle them to display or do something else.

- You need to `try` something .
- To catch the exceptions on what you tried `except`
- Errors catched can be displayed manually with `sys.exc_info()[0]` from the `sys` module

[ ]:
```
# first import the sys module (sys = system)
import sys, json, re
```

Without exception handling the program will crash:

[ ]:
```
# without exceptions the program will crash
print( cake ) # variable cake is not defined, hence it will crash
```

With exception handling you can do something else in case this throws an error:

[ ]:
```
del cake
try:
    print( cake )
except:
    print( 'A problem occurred!')
    print( 'the error is', sys.exc_info()[0])
```

You can specify an action for each different exception type (using their name).

`except NameError` or `except (NameError, ValueError)`

[ ]:
```
try:
    print(cake)
except NameError :
    print( 'name error catched' )
    pass
except ValueError :
    print('value error catched')
    pass
```

Manual error messages can be created with `raise` :

[ ]:
```
print('Enter your group name')
name = input()
if name != 'B2':
    raise ValueError('Not the name I was waiting for!')
```

The keyword `finally` specify something that will always be executed no matter what

[ ]:
```
try:
    print(cake)
except NameError :
    print( 'name error catched' )
    pass
except ValueError :
    print('value error catched')
    pass
finally:
    print( 'goodbye!' )
```

6 Basic Role Playing Game (5) : fine version

- Objective: construct a textual role playing game. The game is only text with choices, conditions to verify the choices and player status.

6.1 Rules

- Player advance from a room to another by textually selecting one of the rooms.
- Player starts with 200 hp (health points)
- A bad decision cost the player to lose 25 hp.
- For each actions, display it from a list of Strings
- Represent user infos as a dictionary
- Use functions to remove code duplicates and reduce the line numbers
- **Add status and health modification function**
- **Access events informations from a json file**
- **Use exceptions to handle input values**
- **Insert the past choices into a log file** at the end of the game

[ ]:
```
def writeDictIntoAFile(filePath, content):
    """ writes a dict into a file. Need the filepath and the content as␣
 ↪arguments."""
    with open(filePath, 'w') as myFile:
        json.dump(content, myFile)

eventsInitial = {
    'first': {
        'message': 'Game Over! Try again '
```

```
    },
    'second': {
        'message': '''Upon opening the door, you can see a huge fest with␣
↪exquisite meals everywhere.
                   Will you eat it?''',
        'paths': {
            'yes': { 'message': 'You ate poison and lost 20 hp', 'hploss': 20},
            'no': { 'message': 'You are still hungry and lose a bit of hp from␣
↪hunger', 'hploss': 10}
        }}
}
writeDictIntoAFile('events.json', eventsInitial) # creates the json file (you␣
↪can do this manually if you work using a local python)
```

[ ]:
```
# Example with dict and  ba sic functions and health management
# Placeholders for you to insert requested code
player = dict()
print("Hello Player One, what's your name?")
player['name'] = input()
player['health'] = 200
print("Welcome", player['name'])

def retrieveEventsFromJson(jsonPath):
    """
    This function reads an external json file containing the events infos and␣
↪returns a dictionary.
    """
    eventsFile = open(jsonPath, 'r') # open the file in 'r' read mode
    eventsDict = json.load(eventsFile) # load the file content into a dict
    eventsFile.close() # do not forget to close the files
    return eventsDict

def writePastChoices( past ):
    """ function that write the past choices to a file .txt """
    pastFile = open('pastChoices.txt', 'a') # we append the choices
    pastFile.write('\n=======\n======\n\n') # append some markers to␣
↪differenciates the runs
    for p in past: # we iterate over the choices to append them
        pastFile.write(p + '\n') # '\n' means new line
    pastFile.close()

def removeHealth(amount):
    ''' Remove some health from the player. And check if he died. '''
    player['health'] -= amount
    if player['health'] <= 0:
        return False
    else:
```

```
        return True



pastChoices = []
events = retrieveEventsFromJson('events.json') # we use the function to get␣
 ↪events dict from the json file



def askAction(event, choices):
    '''
    Ask an action from the user. Stay alive while the action is not recognized.
    '''
    choice = ''
    while choice not in choices:
        print('player status', player)
        choice = input()
        try: # using exception instead of list checking
            print(event[choice]['message']) # if this does not work -> exception
            if 'hploss' in event[choice]:
                alive = removeHealth(event[choice]['hploss'])
                if not alive : break
            pastChoices.append(choice)
            if('paths' in event[choice]):
                askAction(event[choice]['paths'], list(event[choice]['paths'].
 ↪keys()) )
        except:
            print('WRONG : Possible choices', choices )
    return True

print('''Your are in a tiny room. Humidity fills the air but your stomach␣
 ↪reminds you that you are very hungry.
       You are in front of two doors. Behind the first one you can hear muffled␣
 ↪voices.
       Behind the second one you can smell something intriguing.''')
print('Which door do you choose? Type', list(events.keys())[0], 'for the first␣
 ↪room and', list(events.keys())[1],
              'for the second room.')

askAction(events, list(events.keys()) )

print('Here are your remaining stats', player)
writePastChoices( pastChoices ) # we use the function to write the pastChoices␣
 ↪into a file.
```

[ ]: