python-4

September 15, 2020

0.1 # Introduction to Programming – Python

0.2 Course 4 – Exercices and functions

0.2.1 ESSEC Business School

gael.guibon@gmail.com

gael.guibon@telecom-paris.fr

Original content, some parts inspired by Clement Plancq's IM courses

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()`
- Mini textual game

[ ]:
```
# example
user = {
    'name': 'john', # str()
    'age': 19, # int()
    'money': 24.56, # float()
    'job': 'student', # str()
    'friends': ['jack', 'paul'], # list() of two str()
    'stats': {
        'social': 'friendly', # str()
```

```
        'seriousness': 'bad', # str()
        'formerEmployeee': False # bool()
    } # dict()
} # dict()

print(user)
```

2 Nested Dictionaries : exercices

Create a dictionary that represent some cat physical attributes (always print your changes): - Physical attributes - Is it an animal? - How many lives does it have? - How many legs? - What is its maximum age? - Behavior attributes - Does it bites? - What kind of sound does it makes? - Level of agressiveness: low, medium or high

[ ]:
```
# create the cat dictionary here
cat = {
    'physical': {
        'animal': True,
        'lives': 9,
        'legs': 4,
        'maxAge': 15
    },
    'behavior':{
        'bites': False,
        'sound': 'meow',
        'agressiveness': 'low'
    }
}
print(cat)
```

Print the number of lives of this cat:

[ ]:
```
lives = cat['physical']['lives']
print(lives)
```

Print the number of attributes the cat possess:

[ ]:
```
globalAttributes = len(cat.keys())
nestedAttributes = len(cat['physical'].keys()) + len(cat['behavior'].keys())
print('the cat possess', globalAttributes, 'global attributes and',␣
 ↪nestedAttributes, 'nested attributes')
```

Create another dictionary representing some dog attributes: - Physical attributes - Is it an animal? - How many lives does it have? - How many legs? - What is its maximum age? - Behavior attributes - Does it bites? - What kind of sound does it makes? - Level of agressiveness: low, medium or high

[ ]:
```
dog = {
    'physical': {
        'animal': True,
        'lives': 1,
        'legs': 4,
        'maxAge': 11
    },
    'behavior': {
        'bites': True,
        'sound': 'bark',
        'aggressiveness': 'medium'
    }
}
print( dog )
```

Remove the *'animal'* attribute from both the cat and dog dictionaries:

[ ]:
```
print( 'dog before removal', dog)
del dog['physical']['animal']
print( 'dog after removal', dog)
```

2.1 Going deeper in nested dictionaries

Create a dictionary for *livingThings*: two fields: - animals - instances (list) - humans - instances (list)

[ ]:
```
livingThings = {
    'animals': {
        'instances': list()
    },
    'humans': {
        'instances': list()
    }
}

print( livingThings )
```

Insert the list of animals (the cat and the dog dicts) inside a *'instances'* field under the *'animals'* field in the livingThings.

[ ]:
```
livingThings['animals']['instances'].append(dog)
livingThings['animals']['instances'].append(cat)
print( livingThings )
```

Insert the *'averageMaxAge'* in the *'animals'* field. (Use basic math operations for this - see course 1 -)

```
[ ]: # First we intialize a variable to increment the total age from the animal␣
      ↪instances
     totalAge = 0
     # Second we loop over the animal instances to get their 'maxAge' and add it to␣
      ↪the totalAge variable
     for animal in livingThings['animals']['instances']:
         totalAge = totalAge + animal['physical']['maxAge']
     # Third we use basic math operations to compute the average max age and put it␣
      ↪into a 'averageMaxAge' variable
     # total / numberOfElements = basic average here it will be : (11+15) / 2
     # the number of elements is the length of the list of animal instances
     averageMaxAge = totalAge / len( livingThings['animals']['instances'] )
     # Finally we put the value into the desired field of our livingThings dictionary
     livingThings['animals']['averageMaxAge'] = averageMaxAge

     # Now we can print the averageMaxAge
     print( 'average maximum age of animals =',␣
      ↪livingThings['animals']['averageMaxAge'] )

     # and print our whole dictionary
     print( livingThings )
```

Create your neighboor as a dictionary. - What's his/her firstName? - What's his/her lastName? - What are his/her favorite foods? - Does he/she have a sweet tooth? - Does he/she prefer cats or dogs?

```
[ ]: # Now you can easily continue by using your neighboor as a source of infos
```

Create yourself as a dictionary. - What's your firstName? - What's your lastName? - What are your favorite foods? - Do you have a sweet tooth? - Do you prefer cats or dogs?

```
[ ]:
```

Put these two dictionaries into the *'instances'* list.

```
[ ]:
```

Print your *livingThings* dictionary.

```
[ ]:
```

3 Functions

A function is a subprogram, a way to reuse the same code again and again by just calling it.

Functions have: - A name - Arguments (if specified)

Functions do: - A process - Return something (if specified)

Use *docstrings* to documentate functions:

`""" this function does this process and return that value but need those arguments """`

[ ]:
```
def mySuperbFunction(arg1, arg2):
    """
    This superb function needs 2 arguments.
    It returns None (such a shame! this function is still useless)
    """
    return None
```

Examples This function return the sum of the three arguments.

[ ]:
```
def sumItUp(arg1, arg2, arg3):
    """
    Well, this is but a sum of the 3 args.
    """
    return arg1 + arg2 + arg3
```

[ ]:
```
# Now that I defined my function I can use it like this
resultSum = sumItUp(2,3,-5)
print(resultSum)
```

3.1 Functions optional arguments with default value

- Some arguments can have a default value, hence their declaration is optional.
- Mandatory args always comes before optional ones

[ ]:
```
# DEFine the function
def createStudent( age, grade, name='noname'):
    """ This creates a student dict with default name as 'noname' """
    return {'name':name, 'age':age, 'grade':grade}

# now use it
print( createStudent(19, 20) )
print( createStudent( 19, 20, name='superStudent' ) )
# as you can see, the name arg is optional
```

[ ]:
```
# this will not work due to the optional arg placed before a mandatory one
createStudent(name='thisStudent', 19, 20)
```

[ ]:
```
# this will not work due to the lack of mandatory args
createStudent(19)
```

3.2 Arbitrary number of arguments

```
[ ]: def showGroup(groupName, *students, **infos):
         print('group', groupName)
         print('Students:')
         for student in students:
             print(student)
         print('Infos')
         for status, number in infos.items():
             print(status, number)
     showGroup('B2', 'Shuyi', 'Vincent', 'Alizé', 'Mehdi', pythonLovers=2,␣
      ↪superStudents=4)
```

3.3 Variable scope

- Variables defined inside a function possess a local scope (not accessible outside of this function)
- Variables defined outside a function possess a global scope

For instance:

```
[ ]: globalVar = 'I am defined outside a function'
     def doThis(myVar):
         scopedVar = 'I am defined inside a function'
         globalVar = 'yes' # available because it is global
         return True
```

```
[ ]: print(globalVar)
```

```
[ ]: print(scopedVar)
     # it does not work because we are not inside the function where it is defined
```

4 Basic Role Playing Game (4)

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

4.1 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 data as dictionaries or a big nested dict()** (see example below)
- **Use functions** to remove code duplicates and reduce the line numbers

- **Add status and health modification function**

[ ]:
```
# Example with dict and basic functions and health management
player = dict()
print("Hello Player One, what's your name?")
player['name'] = input()
player['health'] = 200
print("Welcome", player['name'])

events = {
    'first': {
        'message': 'Game Over! Try again ' + player['name']
    },
    '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}
        }}
}
pastChoices = []

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

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()
        if choice in choices:
            print(event[choice]['message'])
            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()) )
        else:
            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('You finished playing the game. Here are your past actions :', pastChoices)
print('Here are your remaining stats', player)
```