python crash course 101

Lecture 3 - Functions, Notebooks, Workflow

Who am I?

- Bsc in Statistics
- MSc in Data Science
- Second year of master abroad EPFL
- PhD somewhere

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

SAPIENZA UNIVERSITÀ DI ROMA

For any question feel free to ping me:

cricri.menghini@gmail.com

python crash course 101

Today’s goals

Today's goals

- Few more built-in functions

Today's goals

- Few more built-in functions
- How to create a function

Today's goals

- Few more built-in functions
- How to create a function
- Write a nicely commented Notebook

What is a function?

A function is a block of code used to do something specific.

What is a function?

A function is a block of code used to do something specific.

PROS

- Reusable
- Provides modularity

What is a function?

A function is a block of code used to do something specific.

PROS

- Reusable
- Provides modularity

CONS

- Need to be well-defined
- Requires reasoning before coding

Exploit a simple example to get insights on how to write a function

Task 1: You work in the Human Resources team of Facebook. 100 e-mails should be sent to candidates to tell them if they have been hired or not. So far, you have the list of the name and the interview response.

Exploit a simple example to get insights on how to write a function

Task 1: You work in the Human Resources team of Facebook. 100 e-mails should be sent to candidates to tell them if they have been hired or not. So far, you have the list of the name and the interview response.

1. There are two possible notification to send, a positive and a negative one.
2. The only thing that change both in the negative and positive response is the name of the candidate.

Exploit a simple example to get insights on how to write a function

Task 1: You work in the Human Resources team of Facebook. 100 e-mails should be sent to candidates to tell them if they have been hired or not. So far, you have the list of the name and the interview response.

1. There are two possible notification to send, a positive and a negative one.
2. The only thing that change both in the negative and positive response is the name of the candidate.

Thus, the best thing to do is to automatize the process to write the emails, avoiding to write 100 times the same message.

In general..

When you tackle a problem it is convenient to split it in many small pieces.

In general..

When you tackle a problem it is convenient to split it in many small pieces.

In general..

When you tackle a problem it is convenient to split it in many small pieces.

Solve many little problems is easier than solve a huge task both in terms of **time** and **complexity**!

Define a strategy to accomplish the task

Define a strategy to accomplish the task

Send a response email to each candidate

Define a strategy to accomplish the task

Send a response email to each candidate

Produce the message to send

Define a strategy to accomplish the task

Send a response email to each candidate

Produce the message to send

Positive Response

Negative Response

Build the modules

Where do I start from?

Build the modules

Where do I start from? Start from the **inner** modules

Build the modules

Where do I start from? Start from the **inner** modules

1. Response
2. Produce the message to send
3. Send a response email to each candidate

Response

Response

```python
def positive_response(name):
    """ This function returns the text of the message if the candidate
    is hired.

    Inputs:
    @name: candidate name"""

    print ("""Dear %s, \nWe are happy to inform you that you will be part of our team from now on!
            \t\t\t\t\t\tSee you soon,
            \t\t\t\t\t\tFacebook HR team."""  %name)
```

Response

```
def positive_response(name):
    """ This function returns the text of the message if the candidate
    is hired.

    Inputs:
    @name: candidate name"""

    print ("""Dear %s, \nWe are happy to inform you that you will be part of our team from now on!
            \t\t\t\t\t\tSee you soon,
            \t\t\t\t\t\tFacebook HR team."""  %name)
```

Definition statement: Function name and inputs/arguments

Response

```
def positive_response(name):
    """ This function returns the text of the message if the candidate
    is hired.

    Inputs:
    @name: candidate name"""

    print ("""Dear %s, \nWe are happy to inform you that you will be part of our team from now on!
            \t\t\t\t\t\tSee you soon,
            \t\t\t\t\t\tFacebook HR team."""  %name)
```

Definition statement: Function name and inputs/arguments

DocString: documentation which specifies function functionalities and inputs

Response

```
def positive_response(name):
    """ This function returns the text of the message if the candidate
    is hired.

    Inputs:
    @name: candidate name"""

    print ("""Dear %s, \nWe are happy to inform you that you will be part of our team from now on!
            \t\t\t\t\t\tSee you soon,
            \t\t\t\t\t\tFacebook HR team."""  %name)
```

Definition statement: Function name and inputs/arguments

DocString: documentation which specifies function functionalities and inputs

Body: what the function does

Response (1/2)

```
def positive_response(name):
    """ This function returns the text of the message if the candidate
    is hired.

    Inputs:
    @name: candidate name"""

    print ("""Dear %s, \nWe are happy to inform you that you will be part of our team from now on!
            \t\t\t\t\t\tSee you soon,
            \t\t\t\t\t\tFacebook HR team."""  %name)
```

Definition statement: Function name and inputs/arguments

DocString: documentation which specifies function functionalities and inputs

Body: what the function does

```
positive_response('Dario')
```

```
Dear Dario,
We are happy to inform you that you will be part of our team from now on!
                                                        See you soon,
                                                        Facebook HR team.
```

Response (2/2)

```
def negative_response(name):
    """ This function returns the text of the message if the candidate
    is not hired.

    Inputs:
    @name: candidate name"""

    print ("""Dear %s, \nWe inform you that, unfortunately, you will be part of our team!
Your CV is now in our storage, feel free to continue to look for positio in our company!
            \t\t\t\t\t\tGood luck,
            \t\t\t\t\t\tFacebook HR team."""  %name)
```

Response (2/2)

```
def negative_response(name):
    """ This function returns the text of the message if the candidate
    is not hired.

    Inputs:
    @name: candidate name"""

    print ("""Dear %s, \nWe inform you that, unfortunately, you will be part of our team!
Your CV is now in our storage, feel free to continue to look for positio in our company!
            \t\t\t\t\t\tGood luck,
            \t\t\t\t\t\tFacebook HR team."""  %name)
```

```
negative_response('Giorgio')
```

```
Dear Giorgio,
We inform you that, unfortunately, you will be part of our team!
Your CV is now in our storage, feel free to continue to look for positio in our company!
                                                        Good luck,
                                                        Facebook HR team.
```

Define a strategy to accomplish the task

Send a response email to each candidate

Produce the message to send

Positive Response

Negative Response

```python
def email_response(name, response = 0):
    """ This function returns the text of the message we ahould send to @name.

    It takes as arguments:
    @name: candidate name
    @response: 1 if the response is positive, 0 otherwise (negative response as default)"""


    if response == 0:
        negative_response(name)
        return 'OK'

    elif response == 1:
        positive_response(name)
        return 'OK'

    else:
        return "Error, READ the documentation!"
```

Default value for input variable *response*

```python
def email_response(name, response = 0):
    """ This function returns the text of the message we ahould send to @name.

    It takes as arguments:
    @name: candidate name
    @response: 1 if the response is positive, 0 otherwise (negative response as default)"""


    if response == 0:
        negative_response(name)
        return 'OK'

    elif response == 1:
        positive_response(name)
        return 'OK'

    else:
        return "Error, READ the documentation!"
```

essage to send

```python
def email_response(name, response = 0):
    """ This function returns the text of the message we ahould send to @name.

    It takes as arguments:
    @name: candidate name
    @response: 1 if the response is positive, 0 otherwise (negative response as default)"""


    if response == 0:
        negative_response(name)
        return 'OK'

    elif response == 1:
        positive_response(name)
        return 'OK'

    else:
        return "Error, READ the documentation!"
```

Default value for input variable *response*

Observe how important is the documentation for variable *response*

```python
def email_response(name, response = 0):
    """ This function returns the text of the message we ahould send to @name.

    It takes as arguments:
    @name: candidate name
    @response: 1 if the response is positive, 0 otherwise (negative response as default)"""


    if response == 0:
        negative_response(name)
        return 'OK'

    elif response == 1:
        positive_response(name)
        return 'OK'

    else:
        return "Error, READ the documentation!"
```

Default value for input variable *response*

Observe how important is the documentation for variable *response*

Take into account the possible exception

```python
email_response('Giulio', 1)
```

```
Dear Giulio,
We are happy to inform you that you will be part of our team from now on!
                                                    See you soon,
                                                    Facebook HR team.
```

```python
def email_response(name, response = 0):
    """ This function returns the text of the message we ahould send to @name.

    It takes as arguments:
    @name: candidate name
    @response: 1 if the response is positive, 0 otherwise (negative response as default)"""


    if response == 0:
        negative_response(name)
        return 'OK'

    elif response == 1:
        positive_response(name)
        return 'OK'

    else:
        return "Error, READ the documentation!"
```

Default value for input variable *response*

Observe how important is the documentation for variable *response*

Take into account the possible exception

```python
email_response('Giulio', 1)
```

```
Dear Giulio,
We are happy to inform you that you will be part of our team from now on!
                                                    See you soon,
                                                    Facebook HR team.
```

```python
email_response('Giulio')
```

```
Dear Giulio,
We inform you that, unfortunately, you will not be part of our team!
Your CV is now in our storage, feel free to continue to look for positio in our company!
                                                    Good luck,
                                                    Facebook HR team.
```

Define a strategy to accomplish the task

Send a response email to each candidate

Produce the message to send

Positive Response

Negative Response

Send a response to each candidate (version 1)

```python
def send_email(num_candidates, dict_candidates):
    """
    INPUTS:
    @num_candidates: number of person to send the email
    @dict_candidates: dictionary (key,values):(name, response)

    RETURNS:
    @number_email_sent: number of emails that has been sent
    @wrong_result: list candidate's names the email has not been sent"""

    number_email_sent = 0
    wrong_result = []

    for name, response in dict_candidates.items():

        if email_response(name, response) == 'OK':

            number_email_sent += 1

        else:
            wrong_result.append(name)

    percentage_sent = number_email_sent/num_candidates*100

    return str(percentage_sent) + '%', wrong_result
```

Can you tell me in 10 seconds that it does?

Send a response to each candidate (version 2)

```python
def send_email(num_candidates, dict_candidates):
    """
    INPUTS:
    @num_candidates: number of person to send the email
    @dict_candidates: dictionary (key,values):(name, response)

    RETURNS:
    @number_email_sent: number of emails that has been sent
    @wrong_result: list candidate's names the email has not been sent"""

    # Initialize variables
    number_email_sent = 0
    wrong_result = []

    # For each candidate (and the respective response)
    for name, response in dict_candidates.items():
        # If the text has been correctly created
        if email_response(name, response) == 'OK':
            # Update the number of sent emails
            number_email_sent += 1

        # Otherwise
        else:
            # Append to the list the candidate's name whose email isn't correct
            wrong_result.append(name)

    # Define percentage of sent emails
    percentage_sent = number_email_sent/num_candidates*100

    return str(percentage_sent) + '%', wrong_result
```

COMMENT

Send a respons

```
def send_email(num_cand
    """
    INPUTS:
    @num_candidates: nu
    @dict_candidates: d

    RETURNS:
    @number_email_sent:
    @wrong_result: list

    # Initialize variab
    number_email_sent =
    wrong_result = []

    # For each candidat
    for name, response
        # If the text h
        if email_respon
            # Update th
            number_emai

        # Otherwise
        else:
            # Append to
            wrong_resul

    # Define percentage
    percentage_sent = n

    return str(percenta
```

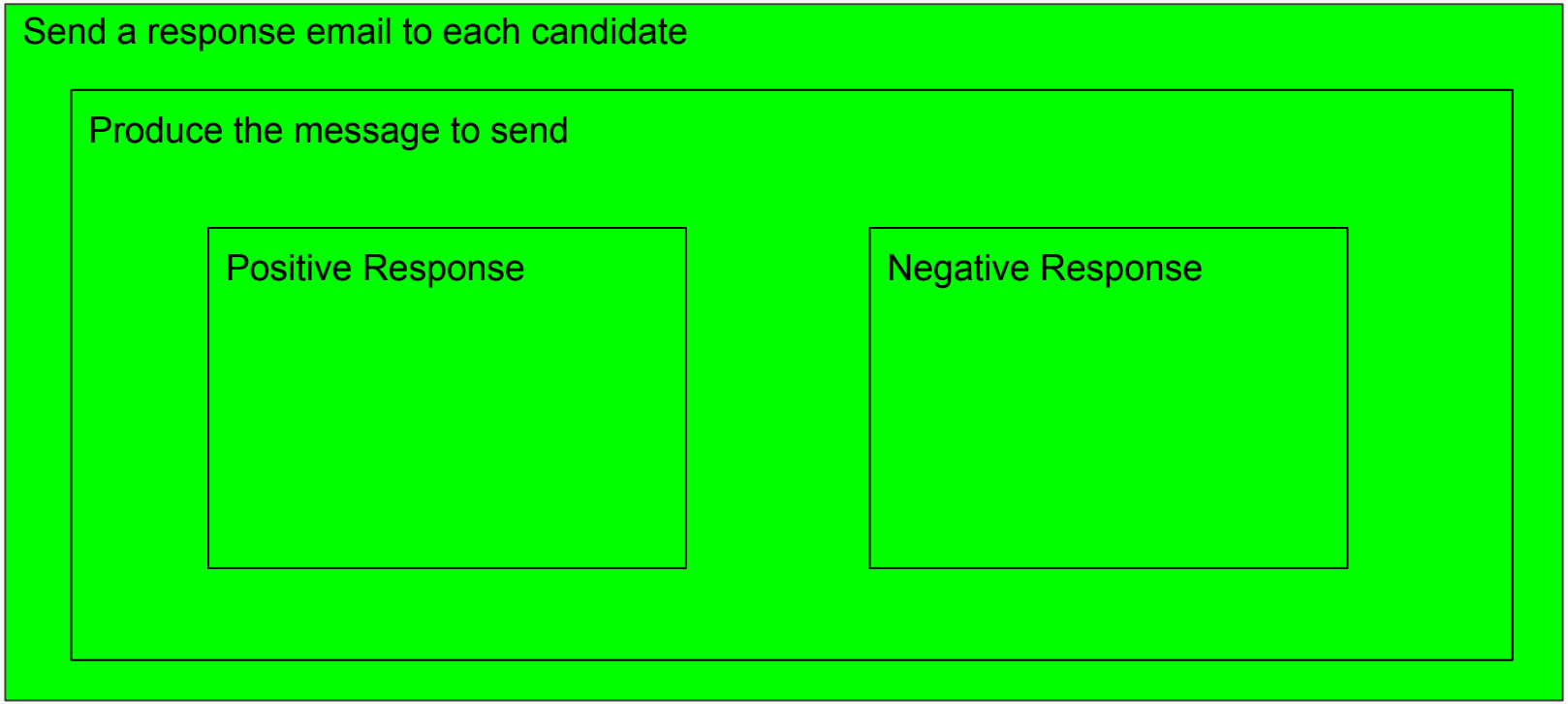
MENT

Send a response to each candidate (version 2)

```python
def send_email(num_candidates, dict_candidates):
    """
    INPUTS:
    @num_candidates: number of person to send the email
    @dict_candidates: dictionary (key,values):(name, response)

    RETURNS:
    @number_email_sent: number of emails that has been sent
    @wrong_result: list candidate's names the email has not been sent"""

    # Initialize variables
    number_email_sent = 0
    wrong_result = []

    # For each candidate (and the respective response)
    for name, response in dict_candidates.items():
        # If the text has been correctly created
        if email_response(name, response) == 'OK':
            # Update the number of sent emails
            number_email_sent += 1

        # Otherwise
        else:
            # Append to the list the candidate's name whose email isn't correct
            wrong_result.append(name)

    # Define percentage of sent emails
    percentage_sent = number_email_sent/num_candidates*100

    return str(percentage_sent) + '%', wrong_result
```

Exploit the exceptions to debug the code

Define a strategy to accomplish the task



REMARKS

1. Give meaningful name to your functions (and its inputs)
2. Provide the documentation for your function (including INPUTS, RETURNS)
3. Comment your code to let the readers have a glance
4. Think about the possible exceptions

Instructions

- Create a directory named “Lectures_Python”

Instructions (1/2)

- Create a directory named “Lectures_Python”
- Go into the directory and clone (using Git), the repository corresponding to this url: https://github.com/Py101/Lectures.git

Instructions (1/2)

- Create a directory named “Lectures_Python”
- Go into the directory and clone (using Git), the repository corresponding to this url: https://github.com/Py101/Lectures.git
- Move to the folder Lectures, then into the folder 03

Instructions (1/2)

- Create a directory named “Lectures_Python”
- Go into the directory and clone (using Git), the repository corresponding to this url: https://github.com/Py101/Lectures.git
- Move to the folder Lectures, then into the folder 03
- Open jupyter notebook

Instructions (1/2)

- Create a directory named “Lectures_Python”
- Go into the directory and clone (using Git), the repository corresponding to this url: https://github.com/Py101/Lectures.git
- Move to the folder Lectures, then into the folder 03
- Open jupyter notebook
- From the home page open the .ipynb file named “Task_2”

Instructions (1/2)

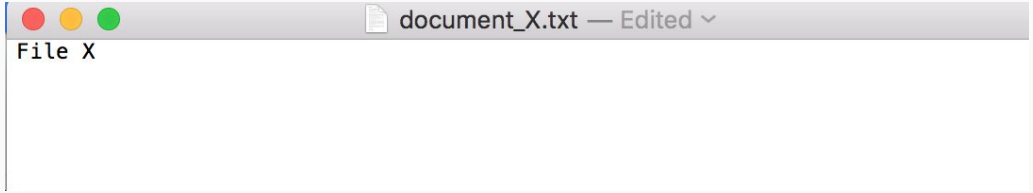
- Create a directory named “Lectures_Python”
- Go into the directory and clone (using Git), the repository corresponding to this url: https://github.com/Py101/Lectures.git
- Move to the folder Lectures, then into the folder 03
- Open jupyter notebook
- From the home page open the .ipynb file named “Task_2”
- Modify the Notebook following the instructions in it (to be more independent I suggest you to open on your laptop the .pdf version of the slides)

It's your turn!

Task 2: We have to create 1000 files and store them in folders (50 per directory).

Deliverables:

- Files: each file is named "document_X.txt", where X is a number from 0 to 999. The content of the file is the following:



- Folders: each folder stores 50 documents and its name defines the files in the directory (e.g. Documents 0-49, Documents 50-99, ...).

How do we proceed? (1/3)

To accomplish the task we need to define different functions.

1. **create_directories(*num_file, file_per_dir*):**

    a. Define the number of folders you obtain with num_file and file_per_dir
    b. Define the list of intervals (*result:* [(0,49), (50,99), ...])
    c. Define the list of directory names (*result:* [“Documents 0-49/”, “Documents 50-99/”, ...])
    d. Create the directories. **Remark:** before creating the folder, check whether it already exists (*hint*: look for it on Google..it’s just one line:-))
    e. Return the list of intervals

How do we proceed? (1/3)

```python
def create_directories(num_file, file_per_dir):
    """The function creates the directories.

    Inputs:
    @num_file: number of total files
    @file_per_dir: number of files per dir

    Returns:
    @intervals: list of intervals each directory covers
    """

    # a. Define the numbers of directory
    [redacted]

    # b. Define the list of intervals ([(0,49), (50, 99), ...])
    [redacted]

    # c. Define the list of directory names (["Documents 0-49", "Documents 50-99", ...])
    [redacted]

    # d. Create the directories (here you need to use the library os.
    # Hint: look for "create a folder checking if it exists python 3" on Google)
    [redacted]

    return intervals
```

How do we proceed? (1/3)

```python
def create_directories(num_file, file_per_dir):
    """The function creates the directories.

    Inputs:
    @num_file: number of total files
    @file_per_dir: number of files per dir

    Returns:
    @intervals: list of intervals each directory covers
    """

    # a. Define the numbers of directory
    num_dir = num_file/file_per_dir

    # b. Define the list of intervals ([(0,49), (50, 99), ...])
    [redacted]

    # c. Define the list of directory names (["Documents 0-49", "Documents 50-99", ...])
    [redacted]

    # d. Create the directories (here you need to use the library os.
    # Hint: look for "create a folder checking if it exists python 3" on Google)
    [redacted]

    return intervals
```

How do we proceed? (1/3)

```python
def create_directories(num_file, file_per_dir):
    """The function creates the directories.

    Inputs:
    @num_file: number of total files
    @file_per_dir: number of files per dir

    Returns:
    @intervals: list of intervals each directory covers
    """

    # a. Define the numbers of directory
    num_dir = num_file/file_per_dir

    # b. Define the list of intervals ([(0,49), (50, 99), ...])
    intervals = [(i, i + (file_per_dir-1)) for i in range(0, num_file, file_per_dir)]

    # c. Define the list of directory names (["Documents 0-49", "Documents 50-99", ...])
    [redacted]

    # d. Create the directories (here you need to use the library os.
    # Hint: look for "create a folder checking if it exists python 3" on Google)
    [redacted]

    return intervals
```

How do we proceed? (1/3)

```python
def create_directories(num_file, file_per_dir):
    """The function creates the directories.

    Inputs:
    @num_file: number of total files
    @file_per_dir: number of files per dir

    Returns:
    @intervals: list of intervals each directory covers
    """

    # a. Define the numbers of directory
    num_dir = num_file/file_per_dir

    # b. Define the list of intervals ([(0,49), (50, 99), ...])
    intervals = [(i, i + (file_per_dir-1)) for i in range(0, num_file, file_per_dir)]

    # c. Define the list of directory names (["Documents 0-49", "Documents 50-99", ...])
    directories = ['Documents ' + '-'.join([str(i),str(j)]) + '/' for i,j in intervals]

    # d. Create the directories (here you need to use the library os.
    # Hint: look for "create a folder checking if it exists python 3" on Google)
    [redacted]

    return intervals
```

How do we proceed? (1/3)

```python
def create_directories(num_file, file_per_dir):
    """The function creates the directories.

    Inputs:
    @num_file: number of total files
    @file_per_dir: number of files per dir

    Returns:
    @intervals: list of intervals each directory covers
    """

    # a. Define the numbers of directory
    num_dir = num_file/file_per_dir

    # b. Define the list of intervals ([(0,49), (50, 99), ...])
    intervals = [(i, i + (file_per_dir-1)) for i in range(0, num_file, file_per_dir)]

    # c. Define the list of directory names (["Documents 0-49", "Documents 50-99", ...])
    directories = ['Documents ' + '-'.join([str(i),str(j)]) + '/' for i,j in intervals]

    # d. Create the directories (here you need to use the library os.
    # Hint: look for "create a folder checking if it exists python 3" on Google)
    for directory in directories:
        if not os.path.exists(directory):
            os.makedirs(directory)

    return intervals
```

How do we proceed? (2/3)

2. **create_file(*dir*, *idx*)**:

   a. Open a new file where you can write
   b. Write the content “File X” in the file, where X is the *idx* of the file
   c. Save the file in the correct directory (*dir*) with file’s name “document_X”, where X is again *idx*

How do we proceed? (2/3)

```python
def create_file(direct, idx):
    """The function creates a file and stores it in the right directory.

    Inputs:
    @direct: directory path (e.g. 'Documents 0-49/')
    @idx: index (number) of the file

    """

    # a. Open a new file where you can write in the direct and Write the content
```

How do we proceed? (2/3)

```python
def create_file(direct, idx):
    """The function creates a file and stores it in the right directory.

    Inputs:
    @direct: directory path (e.g. 'Documents 0-49/')
    @idx: index (number) of the file

    """

    # a. Open a new file where you can write in the direct and Write the content
    with open(direct + 'document_' + str(idx), 'w') as f:
        f.write('File ' + str(idx) + '.txt')
```

How do we proceed? (3/3)

3. **solve_task(*num_files*, *intervals*)**:

    a. For each index (from 0 to 999)

        b. For each interval check if the index is in the interval

            c. If yes, create the file and break the loop

How do we proceed? (3/3)

```python
def solve_task(num_file, intervals):
    """The function creates the input number of files storing them in the correct folders.

    Inputs:
    @num_file: number of total files
    @intervals: list of intervals each directory covers
    """

    # a. for each index from 0 to 999
    [redacted]
        # b. for each interval, check if the index is in the interval
        [redacted]
            # c. if yes
            [redacted]
                # create the file and break
                [redacted]
```

How do we proceed? (3/3)

```python
def solve_task(num_file, intervals):
    """The function creates the input number of files storing them in the correct folders.

    Inputs:
    @num_file: number of total files
    @intervals: list of intervals each directory covers
    """

    # a. for each index from 0 to 999
    for idx in range(1000):
        # b. for each interval, check if the index is in the interval
        [redacted]
            # c. if yes
            [redacted]
                # create the file and break
                [redacted]
```

How do we proceed? (3/3)

```python
def solve_task(num_file, intervals):
    """The function creates the input number of files storing them in the correct folders.

    Inputs:
    @num_file: number of total files
    @intervals: list of intervals each directory covers
    """

    # a. for each index from 0 to 999
    for idx in range(1000):
        # b. for each interval, check if the index is in the interval
        for ep1, ep2 in intervals:
            # c. if yes
            [redacted]
                # create the file and break
                [redacted]
```

How do we proceed? (3/3)

```python
def solve_task(num_file, intervals):
    """The function creates the input number of files storing them in the correct folders.

    Inputs:
    @num_file: number of total files
    @intervals: list of intervals each directory covers
    """

    # a. for each index from 0 to 999
    for idx in range(1000):
        # b. for each interval, check if the index is in the interval
        for ep1, ep2 in intervals:
            # c. if yes
            if ep1 <= idx <= ep2:
                # create the file and break
```

How do we proceed? (3/3)

```python
def solve_task(num_file, intervals):
    """The function creates the input number of files storing them in the correct folders.

    Inputs:
    @num_file: number of total files
    @intervals: list of intervals each directory covers
    """

    # a. for each index from 0 to 999
    for idx in range(1000):
        # b. for each interval, check if the index is in the interval
        for ep1, ep2 in intervals:
            # c. if yes
            if ep1 <= idx <= ep2:
                # create the file and break
                create_file('Documents ' + str(ep1) + '-' + str(ep2) + '/', idx)
                break
```

CONGRATS!

101
python crash course

Few essential built-in functions

What do I want to do?	Function
I need to use a function and use it only once	
I want to apply the same function to each element of a list	
Given a list, and a function which defines a condition (e.g. x > 0), I want back the elements of the list for which the condition is true	
Given a list and a function, I want to apply the latter to sequential pairs of variable is the list	

Few essential built-in functions

What do I want to do?	Function
I need to use a function and use it only once	**lambda**
I want to apply the same function to each element of a list	
Given a list, and a function which defines a condition (e.g. x > 0), I want back the elements of the list for which the condition is true	
Given a list and a function, I want to apply the latter to sequential pairs of variable is the list	

Few essential built-in functions

What do I want to do?	Function
I need to use a function and use it only once	**lambda**
I want to apply the same function to each element of a list	**map**
Given a list, and a function which defines a condition (e.g. x > 0), I want back the elements of the list for which the condition is true	
Given a list and a function, I want to apply the latter to sequential pairs of variable is the list	

Few essential built-in functions

What do I want to do?	Function
I need to use a function and use it only once	**lambda**
I want to apply the same function to each element of a list	**map**
Given a list, and a function which defines a condition (e.g. x > 0), I want back the elements of the list for which the condition is true	**filter**
Given a list and a function, I want to apply the latter to sequential pairs of variable is the list	

Few essential built-in functions

What do I want to do?	Function
I need to use a function and use it only once	**lambda**
I want to apply the same function to each element of a list	**map**
Given a list, and a function which defines a condition (e.g. x > 0), I want back the elements of the list for which the condition is true	**filter**
Given a list and a function, I want to apply the latter to sequential pairs of variable in the list	**reduce**

Beloved lambda

```python
def squared_num(x):
    """ The function returns the squared of a number
    Inputs:
    @x: number
    """
    return x**2
```

Beloved lambda

```python
def squared_num(x):
    """ The function returns the squared of a number
    Inputs:
    @x: number
    """
    return x**2
```

Function which does one operation and is written in **too many** lines

Beloved lambda

```python
def squared_num(x):
    """ The function returns the squared of a number
    Inputs:
    @x: number
    """
    return x**2
```

Function which does one operation and is written in **too many** lines

```python
squared = lambda x: x**2
```

Beloved lambda

```python
def squared_num(x):
    """ The function returns the squared of a number
    Inputs:
    @x: number
    """
    return x**2
```

Function which does one operation and is written in **too many** lines

```python
squared = lambda x: x**2
```

Exactly the same function but written in **fewer words and lines**

Beloved lambda

```python
def squared_num(x):
    """ The function returns the squared of a number
    Inputs:
    @x: number
    """
    return x**2
```

Function which does one operation and is written in **too many** lines

```python
squared = lambda x: x**2
```

Exactly the same function but written in **fewer words and lines**

Combo Map + lambda

```
list_num = [1,2,3,4,5,6,7,8,9,10]
```

Combo Map + lambda

```
list_num = [1,2,3,4,5,6,7,8,9,10]
```

We want each item of the list to be divided by 2

Combo Map + lambda

```
list_num = [1,2,3,4,5,6,7,8,9,10]
```

We want each item of the list to be divided by 2

map + lambda

Combo Map + lambda

```
list_num = [1,2,3,4,5,6,7,8,9,10]
```

We want each item of the list to be divided by 2

map + lambda

```
list(map(lambda x: x/2, list_num))
```

```
[0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]
```

Combo Filter + lambda

```
list_num = [1,2,3,4,5,6,7,8,9,10]
```

Combo Filter + lambda

```
list_num = [1,2,3,4,5,6,7,8,9,10]
```

We want the list of items greater than 5

Combo Filter + lambda

```
list_num = [1,2,3,4,5,6,7,8,9,10]
```

We want the list of items greater than 5

filter + lambda

Combo Filter + lambda

```
list_num = [1,2,3,4,5,6,7,8,9,10]
```

We want the list of items greater than 5

filter + lambda

```
list(filter(lambda x: x>5, list_num))
```

```
[6, 7, 8, 9, 10]
```

Combo Reduce + lambda

```
list_num = [1,2,3,4,5,6,7,8,9,10]
```

Combo Reduce + lambda

```
list_num = [1,2,3,4,5,6,7,8,9,10]
```

We want to apply the following function:

```
function_lambda = lambda x,y: x+y
```

Combo Reduce + lambda

```
list_num = [1,2,3,4,5,6,7,8,9,10]
```

We want to apply the following function:

```
function_lambda = lambda x,y: x+y
```

```
from functools import reduce
```

```
reduce(function_lambda, list_num)
```

```
55
```

Combo Reduce + lambda

```
list_num = [1,2,3,4,5,6,7,8,9,10]
```

We want to apply the following function:

```
function_lambda = lambda x,y: x+y
```

```
from functools import reduce
```

```
reduce(function_lambda, list_num)
```

```
55
```

Combo Reduce + lambda

```
list_num = [1,2,3,4,5,6,7,8,9,10]
```

We want to apply the following function:

```
function_lambda = lambda x,y: x+y
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
from functools import reduce
```

```
reduce(function_lambda, list_num)
```

```
55
```

Combo Reduce + lambda

```
list_num = [1,2,3,4,5,6,7,8,9,10]
```

We want to apply the following function:

```
function_lambda = lambda x,y: x+y
```

```
from functools import reduce
```

```
reduce(function_lambda, list_num)
```

```
55
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
[3, 3, 4, 5, 6, 7, 8, 9, 10]
```

Combo Reduce + lambda

```
list_num = [1,2,3,4,5,6,7,8,9,10]
```

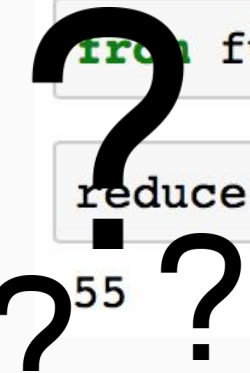
We want to apply the following function:

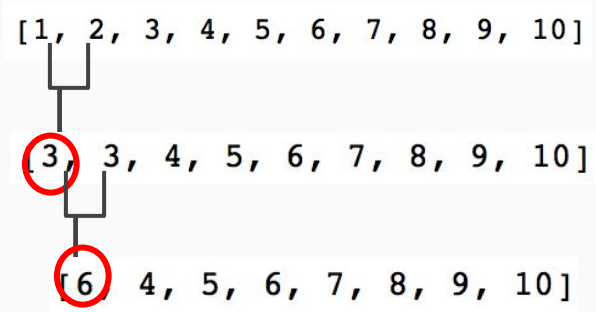
```
function_lambda = lambda x,y: x+y
```

```
from functools import reduce
```

```
reduce(function_lambda, list_num)
```

```
55
```





Combo Reduce + lambda

```
list_num = [1,2,3,4,5,6,7,8,9,10]
```

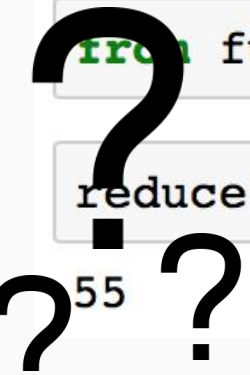
We want to apply the following function:

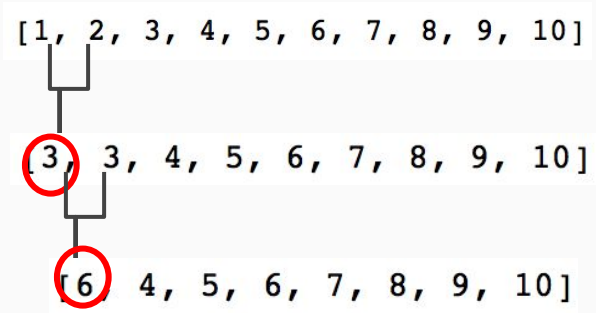
```
function_lambda = lambda x,y: x+y
```

```
from functools import reduce
```

```
reduce(function_lambda, list_num)
```

```
55
```





Until 55!!

LET’S DO SOME EXERCISES!