# Cài đặt môi trường – sử dụng

## Chạy python trên SublimeText

Ctrl + Shift + B : lần đầu

Ctrl + B: các lần sau

## Cài đặt package Anaconda để viết Python trên SublimeText thuận tiện

Ctrl + Shift + P -> Install Package -> Anaconda

## Sử dụng pip – để cài các package cho Python (dạng như Package Control)

(nếu chưa cài thì cài pip)

pip install <tên gói>

Các gói thông dụng:

- random: pip install random #hinh nhu python 3.4 da da co san goi nay

- requests: pip install requests

- beautifulsoup4: pip install beautifulsoup4

# Một số đặc tính Python so với Java, C++

## Thuộc tính của lớp và của đối tượng

Thuộc tính của lớp giống thuộc tính bình thường trong Java, C++ là khai báo trong lớp (ngoài các hàm), còn thuộc tính của đối tượng của Python thì nằm trong hàm (Java, C++ không phân biệt vấn đề này)

class Girl():

gender = ‘female’

def \_\_init\_\_(self, name):

self.name = name

def name(self)

print(‘Name: ‘ + self.name)

lisa = Girl(‘Lisa’)

mel = Girl(‘Melisa’)

lisa.name()

mel.name()

## Đa thừa kế

Khác với các ngôn ngữ như Java, C++,… Python cho phép đa thừa kế

class Mario():

def move(self):

print(‘I am moving’)

class Shroom():

def eat\_shroom(self):

print(‘Now I am bigger!’)

class BigMario(Mario, Shroom):

pass

bm = BigMario()

bm.move()

bm.eat\_shroom

# Code mẫu

## Download ảnh từ web

Code mẫu

import random

import urllib.request

def download\_web\_img(url):

name = random.randrange(1,2000)

fullname = str(name) + ".jpg"

urllib.request.urlretrieve(url, fullname)

download\_web\_img('https://vnn-imgs-f.vgcloud.vn/2018/09/18/13/an-tuong-hinh-anh-ngoai-giao-xe-hoi-moon-kim-tren-duong-pho-binh-nhuong.jpg')

## Read and write file

fw = open('test.txt', 'w')

fw.write('Bui Cong Thanh\n')

fw.write('Xin chao ban!')

fw.close()

fr = open('test.txt', 'r')

text = fr.read()

print(text)

fr.close()

## Download files from web

## Guess the type

To find out the type of a value or a variable that refers to that value, you can use the [type()](https://docs.python.org/3/library/functions.html#type) function. Suppose you've defined a variable a, but you forgot the type of this variable. To determine the type of a, simply execute:

type(a)

## Operations with other types

Filip mentioned that different types behave differently in Python.

When you sum two strings, for example, you'll get different behavior than when you sum two integers or two booleans.

In the script some variables with different types have already been created. It's up to you to use them.

##### Instructions

100 XP

* Calculate the product of savings and factor. Store the result in year1.
* What do you think the resulting type will be? Find out by printing out the type of year1.
* Calculate the sum of desc and desc and store the result in a new variable doubledesc.
* Print out doubledesc. Did you expect this?

savings = 100

factor = 1.1

desc = "compound interest"

# Assign product of factor and savings to year1

# Print the type of year1

# Python List

## Create a list

As opposed to int, bool etc., a list is a **compound data type**; you can group values together:

a = "is"

b = "nice"

my\_list = ["my", "list", a, b]

After measuring the height of your family, you decide to collect some information on the house you're living in. The areas of the different parts of your house are stored in separate variables for now, as shown in the script.

##### Instructions

100 XP

* Create a list, areas, that contains the area of the hallway (hall), kitchen (kit), living room (liv), bedroom (bed) and bathroom (bath), in this order. Use the predefined variables.
* Print areas with the [print()](https://docs.python.org/3/library/functions.html#print) function.

# area variables (in square meters)

hall = 11.25

kit = 18.0

liv = 20.0

bed = 10.75

bath = 9.50

# Create list areas

## Create list with different types

A list can contain any Python type. Although it's not really common, a list can also contain a mix of Python types including strings, floats, booleans, etc.

The printout of the previous exercise wasn't really satisfying. It's just a list of numbers representing the areas, but you can't tell which area corresponds to which part of your house.

The code on the right is the start of a solution. For some of the areas, the name of the corresponding room is already placed in front. Pay attention here! "bathroom" is a string, while bath is a variable that represents the float 9.50 you specified earlier.

hat represents the float 9.50 you specified earlier.

### Instructions

100 XP

* Finish the line of code that creates the areas list. Build the list so that the list first contains the name of each room as a string and then its area. In other words, add the strings "hallway", "kitchen" and "bedroom" at the appropriate locations.
* Print areas again; is the printout more informative this time?

# area variables (in square meters)

hall = 11.25

kit = 18.0

liv = 20.0

bed = 10.75

bath = 9.50

# Adapt list areas

areas = [hall, kit, "living room", liv, bed, "bathroom", bath]

# Print areas

## Select the valid list

A list can contain any Python type. But a list itself is also a Python type. That means that a list can also contain a list! Python is getting funkier by the minute, but fear not, just remember the list syntax:

my\_list = [el1, el2, el3]

Can you tell which ones of the following lines of Python code are valid ways to build a list?

A. [1, 3, 4, 2] B. [[1, 2, 3], [4, 5, 7]] C. [1 + 2, "a" \* 5, 3]

## List of lists

As a data scientist, you'll often be dealing with a lot of data, and it will make sense to group some of this data.

Instead of creating a flat list containing strings and floats, representing the names and areas of the rooms in your house, you can create a list of lists. The script on the right can already give you an idea.

Don't get confused here: "hallway" is a string, while hall is a variable that represents the float 11.25 you specified earlier.

##### Instructions

100 XP

* Finish the list of lists so that it also contains the bedroom and bathroom data. Make sure you enter these in order!
* Print out house; does this way of structuring your data make more sense?
* Print out the type of house. Are you still dealing with a list?

# area variables (in square meters)

hall = 11.25

kit = 18.0

liv = 20.0

bed = 10.75

bath = 9.50

# house information as list of lists

house = [["hallway", hall],

["kitchen", kit],

["living room", liv]]

# Print out house

# Print out the type of house

## Subset and conquer

Subsetting Python lists is a piece of cake. Take the code sample below, which creates a list x and then selects "b" from it. Remember that this is the second element, so it has index 1. You can also use negative indexing.

x = ["a", "b", "c", "d"]

x[1]

x[-3] # same result!

Remember the areas list from before, containing both strings and floats? Its definition is already in the script. Can you add the correct code to do some Python subsetting?

##### Instructions

100 XP

* Print out the second element from the areas list, so 11.25.
* Subset and print out the last element of areas, being 9.50. Using a negative index makes sense here!
* Select the number representing the area of the living room and print it out.

# Create the areas list

areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

# Print out second element from areas

print(areas[1])

# Print out last element from areas

print(areas[-1])

# Print out the area of the living room

print(areas[\_])

## Subset and calculate

After you've extracted values from a list, you can use them to perform additional calculations. Take this example, where the second and fourth element of a list x are extracted. The strings that result are pasted together using the + operator:

x = ["a", "b", "c", "d"]

print(x[1] + x[3])

##### Instructions

100 XP

* Using a combination of list subsetting and variable assignment, create a new variable, eat\_sleep\_area, that contains the sum of the area of the kitchen and the area of the bedroom.
* Print the new variable eat\_sleep\_area.

# Create the areas list

areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

# Sum of kitchen and bedroom area: eat\_sleep\_area

# Print the variable eat\_sleep\_area

# Slicing and dicing

Selecting single values from a list is just one part of the story. It's also possible to slice your list, which means selecting multiple elements from your list. Use the following syntax:

my\_list[start:end]

The start index will be included, while the end index is not.

The code sample below shows an example. A list with "b" and "c", corresponding to indexes 1 and 2, are selected from a list x:

x = ["a", "b", "c", "d"]

x[1:3]

The elements with index 1 and 2 are included, while the element with index 3 is not.

##### Instructions

100 XP

* Use slicing to create a list, downstairs, that contains the first 6 elements of areas.
* Do a similar thing to create a new variable, upstairs, that contains the last 4 elements of areas.
* Print both downstairs and upstairs using [print()](https://docs.python.org/3/library/functions.html#print).

# Create the areas list

areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

# Use slicing to create downstairs

# Use slicing to create upstairs

# Print out downstairs and upstairs

## Slicing and dicing (2)

In the video, Filip first discussed the syntax where you specify both where to begin and end the slice of your list:

my\_list[begin:end]

However, it's also possible not to specify these indexes. If you don't specify the begin index, Python figures out that you want to start your slice at the beginning of your list. If you don't specify the end index, the slice will go all the way to the last element of your list. To experiment with this, try the following commands in the IPython Shell:

x = ["a", "b", "c", "d"]

x[:2]

x[2:]

x[:]

##### Instructions

100 XP

Use slicing to create the lists downstairs and upstairs again, but this time without using indexes if it's not necessary. Remember downstairs is the first 6 elements of areas and upstairs is the last 4 elements of areas

# Create the areas list

areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

# Alternative slicing to create downstairs

# Alternative slicing to create upstairs

## Subsetting lists of lists

You saw before that a Python list can contain practically anything; even other lists! To subset lists of lists, you can use the same technique as before: square brackets. Try out the commands in the following code sample in the IPython Shell:

x = [["a", "b", "c"],

["d", "e", "f"],

["g", "h", "i"]]

x[2][0]

x[2][:2]

x[2] results in a list, that you can subset again by adding additional square brackets.

What will house[-1][1] return? house, the list of lists that you created before, is already defined for you in the workspace. You can experiment with it in the IPython Shell.

## Replace list elements

Replacing list elements is pretty easy. Simply subset the list and assign new values to the subset. You can select single elements or you can change entire list slices at once.

Use the IPython Shell to experiment with the commands below. Can you tell what's happening and why?

x = ["a", "b", "c", "d"]

x[1] = "r"

x[2:] = ["s", "t"]

For this and the following exercises, you'll continue working on the areas list that contains the names and areas of different rooms in a house.

##### Instructions

100 XP

* Update the area of the bathroom area to be 10.50 square meters instead of 9.50.
* Make the areas list more trendy! Change "living room" to "chill zone".

# Create the areas list

areas = ["hallway", 11.25, "kitchen", 18.0, "living room", 20.0, "bedroom", 10.75, "bathroom", 9.50]

# Correct the bathroom area

# Change "living room" to "chill zone"

## Extend a list

If you can change elements in a list, you sure want to be able to add elements to it, right? You can use the + operator:

x = ["a", "b", "c", "d"]

y = x + ["e", "f"]

You just won the lottery, awesome! You decide to build a poolhouse and a garage. Can you add the information to the areas list?

##### Instructions

100 XP

* Use the + operator to paste the list ["poolhouse", 24.5] to the end of the areas list. Store the resulting list as areas\_1.
* Further extend areas\_1 by adding data on your garage. Add the string "garage" and float 15.45. Name the resulting list areas\_2.

# Create the areas list and make some changes

areas = ["hallway", 11.25, "kitchen", 18.0, "chill zone", 20.0,

"bedroom", 10.75, "bathroom", 10.50]

# Add poolhouse data to areas, new list is areas\_1

# Add garage data to areas\_1, new list is areas\_2

## Delete list elements

Finally, you can also remove elements from your list. You can do this with the del statement:

x = ["a", "b", "c", "d"]

del(x[1])

**Pay attention here: as soon as you remove an element from a list, the indexes of the elements that come after the deleted element all change!**

The updated and extended version of areas that you've built in the previous exercises is coded below. You can copy and paste this into the IPython Shell to play around with the result.

areas = ["hallway", 11.25, "kitchen", 18.0,

"chill zone", 20.0, "bedroom", 10.75,

"bathroom", 10.50, "poolhouse", 24.5,

"garage", 15.45]

There was a mistake! The amount you won with the lottery is not that big after all and it looks like the poolhouse isn't going to happen. You decide to remove the corresponding string and float from the areas list.

The ; sign is used to place commands on the same line. The following two code chunks are equivalent:

# Same line

command1; command2

# Separate lines

command1

command2

Which of the code chunks will do the job for us?

(xóa 1 lúc 2 phần tử 3 và 4 khác xóa phần tử 3 và sau đó phần tử 4 vì trật tự đã thay đổi)

## Inner workings of lists

At the end of the video, Filip explained how Python lists work behind the scenes. In this exercise you'll get some hands-on experience with this.

The Python code in the script already creates a list with the name areas and a copy named areas\_copy. Next, the first element in the areas\_copy list is changed and the areas list is printed out. If you hit Run Code you'll see that, although you've changed areas\_copy, the change also takes effect in the areas list. That's because areas and areas\_copy point to the same list.

If you want to prevent changes in areas\_copy from also taking effect in areas, you'll have to do a more explicit copy of the areas list. You can do this with [list()](https://docs.python.org/3/library/functions.html#func-list) or by using [:].

##### Instructions

100 XP

Change the second command, that creates the variable areas\_copy, such that areas\_copy is an explicit copy of areas. After your edit, changes made to areas\_copy shouldn't affect areas. Hit Submit Answer to check this.

# Create list areas

areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Create areas\_copy

areas\_copy = areas

# Change areas\_copy

areas\_copy[0] = 5.0

# Print areas

print(areas)

## Selective import

General imports, like import math, make **all** functionality from the math package available to you. However, if you decide to only use a specific part of a package, you can always make your import more selective:

from math import pi

Let's say the Moon's orbit around planet Earth is a perfect circle, with a radius r (in km) that is defined in the script.

##### Instructions

100 XP

* Perform a selective import from the math package where you only import the radians function.
* Calculate the distance travelled by the Moon over 12 degrees of its orbit. Assign the result to dist. You can calculate this as r \* phi, where r is the radius and phi is the angle in radians. To convert an angle in degrees to an angle in radians, use the [radians()](https://docs.python.org/3/library/math.html#math.radians) function, which you just imported.
* Print out dist.

# Definition of radius

r = 192500

# Import radians function of math package

# Travel distance of Moon over 12 degrees. Store in dist.

# Print out dist

## Different ways of importing

There are several ways to import packages and modules into Python. Depending on the import call, you'll have to use different Python code.

Suppose you want to use the function [inv()](http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.linalg.inv.html), which is in the linalg subpackage of the scipy package. You want to be able to use this function as follows:

my\_inv([[1,2], [3,4]])

Which import statement will you need in order to run the above code without an error?

import scipy

import scipy.linalg

from scipy.linalg import my\_inv

from scipy.linalg import inv as my\_inv

## Your First NumPy Array

In this chapter, we're going to dive into the world of baseball. Along the way, you'll get comfortable with the basics of numpy, a powerful package to do data science.

A list baseball has already been defined in the Python script, representing the height of some baseball players in centimeters. Can you add some code here and there to create a numpy array from it?

##### Instructions

100 XP

* Import the numpy package as np, so that you can refer to numpy with np.
* Use [np.array()](http://docs.scipy.org/doc/numpy-1.10.0/glossary.html#term-array) to create a numpy array from baseball. Name this array np\_baseball.
* Print out the type of np\_baseball to check that you got it right.

# Create list baseball

baseball = [180, 215, 210, 210, 188, 176, 209, 200]

# Import the numpy package as np

# Create a numpy array from baseball: np\_baseball

# Print out type of np\_baseball

## Baseball players' height

You are a huge baseball fan. You decide to call the MLB (Major League Baseball) and ask around for some more statistics on the height of the main players. They pass along data on more than a thousand players, which is stored as a regular Python list: height. The height is expressed in inches. Can you make a numpy array out of it and convert the units to meters?

height is already available and the numpy package is loaded, so you can start straight away (Source: [stat.ucla.edu](http://wiki.stat.ucla.edu/socr/index.php/SOCR_Data_MLB_HeightsWeights)).

##### Instructions

100 XP

* Create a numpy array from height. Name this new array np\_height.
* Print np\_height.
* Multiply np\_height with 0.0254 to convert all height measurements from inches to meters. Store the new values in a new array, np\_height\_m.
* Print out np\_height\_m and check if the output makes sense.

# height is available as a regular list

# Import numpy

import numpy as np

# Create a numpy array from height: np\_height

# Print out np\_height

# Convert np\_height to m: np\_height\_m

# Print np\_height\_m

## Baseball player's BMI

The MLB also offers to let you analyze their weight data. Again, both are available as regular Python lists: height and weight. height is in inches and weight is in pounds.

It's now possible to calculate the BMI of each baseball player. Python code to convert height to a numpy array with the correct units is already available in the workspace. Follow the instructions step by step and finish the game!

##### Instructions

100 XP

* Create a numpy array from the weight list with the correct units. Multiply by 0.453592 to go from pounds to kilograms. Store the resulting numpy array as np\_weight\_kg.
* Use np\_height\_m and np\_weight\_kg to calculate the BMI of each player. Use the following equation:

BMI=weight(kg) / height(m)2

 Save the resulting numpy array as bmi.

 Print out bmi.

# height and weight are available as regular lists

# Import numpy

import numpy as np

# Create array from height with correct units: np\_height\_m

np\_height\_m = np.array(height) \* 0.0254

# Create array from weight with correct units: np\_weight\_kg

# Calculate the BMI: bmi

# Print out bmi

## Lightweight baseball players

To subset both regular Python lists and numpy arrays, you can use square brackets:

x = [4 , 9 , 6, 3, 1]

x[1]

import numpy as np

y = np.array(x)

y[1]

For numpy specifically, you can also use boolean numpy arrays:

high = y > 5

y[high]

The code that calculates the BMI of all baseball players is already included. Follow the instructions and reveal interesting things from the data!

##### Instructions

100 XP

* Create a boolean numpy array: the element of the array should be True if the corresponding baseball player's BMI is below 21. You can use the < operator for this. Name the array light.
* Print the array light.
* Print out a numpy array with the BMIs of all baseball players whose BMI is below 21. Use light inside square brackets to do a selection on the bmi array.

# height and weight are available as a regular lists

# Import numpy

import numpy as np

# Calculate the BMI: bmi

np\_height\_m = np.array(height) \* 0.0254

np\_weight\_kg = np.array(weight) \* 0.453592

bmi = np\_weight\_kg / np\_height\_m \*\* 2

# Create the light array

# Print out light

# Print out BMIs of all baseball players whose BMI is below 21

## Unpacking aguments

def myfunction(arg1, arg2, arg3)

res = arg1 + arg2 + arg3

print (res)

values = [“Bui”, “Cong”, “Thanh”]

myfunction(values[0], values[1], values[2])

myfunction(\*values) #ket qua tuong tu nhu tren

## Dictonary

classmates = {'Tony':'Tall but lazy', 'Emma': 'Smart and Urly', 'Luzy': 'Beautiful but Duty'}

print(classmates)

print(classmates['Emma'])

for k, v in classmates.items():

print(k + " - " + v)

## Modules

## Threading

Một chương trình thông thường chạy tuần tự các lệnh, ví dụ chương trình gồm 2 lệnh

chucnang1()

chucnang2()

thì chucnang1() thực hiện xong mới thực hiện chức năng 2.

Tuy nhiên điều này không hợp lý trong thực tế, ví dụ chương trình đang xử lý chức năng bên dưới mà không cho phép người dùng thao tác với giao diện sẽ không thân thiện với người dùng (nhất là dữ liệu lớn). Vậy làm cách nào có thể một chương trình chạy 1 lúc nhiều chức năng? Để giải quyết người ta đưa vào khái niệm threading – chia chương trình gồm nhiều chức năng song song.

Đối với ví dụ trên thì nếu dùng threading thì chức năng 2 chạy song song với chức năng 1 chứ không đợi khi kết thúc chức năng 1 mới chạy.

import threading

class ExampleMessenger(threading.Thread):

def run(self):

for \_ in range(10):

print(threading.currentThread().getName())

x = ExampleMessenger(name=”Send out message”)

y = ExampleMessenger(name=”Receive message”)

x.start()

y.start()

Khi chạy sẽ cho kết quả “send…” và “receive…” lẫn vào nhau chứ không phải “send…” xong mới “receiv…”

## Extracting a list to tuble

VD:

hocsinh = [‘ms01’, ‘nguyen van an’, ‘2000’, ‘Quang Ngai’]

Để truy suất các phần tử của list này thì: hocsinh[0], hocsinh[1]…..

Có cách truy suất tiện hơn là dùng tuble

ma, ten, namsinh, quequan = hocsinh

Lúc này sẽ hiểu tương ứng theo vị trí. Thông thường tuble sẽ có kích thước bằng kích thước của danh sách. Nếu không thì thường là đầu, giữa và cuối, với giữa có dấu \* để cho biết kích thước không cố định.

VD: dau, \*giua, cuoi = [11, 22, 2333, 22, 98, 23]

Với ví dụ này, dau sẽ có giá trị 11, cuối có giá trị 23 còn giua là mảng 22, 2333, 22, 98

## ZIP

Thường dùng để nối các phần tử tương ứng của 2 hay nhiều mảng với cùng kích thước.

first = [“Le”, “Tran, “Nguyen”, “Dinh”, “Bui”]

last = [“An”, “Dung”, “Tuan”, “Oc”, “Cong”]

names = zip(first, last)

for ho, ten in names:

print(ho, ten)

## Map

Để chuyển 1 danh sách thành 1 danh sách mới thông qua một hàm được định nghĩa. Map sẽ thực hiện việc qua từng phần tử của mảng gốc và ánh xạ thành giá trị mới thông qua hàm được định nghĩa.

VD

income = [10, 20, 30]

def doubleincome(money):

return money \* 2

# duyet qua tung phan tu cua income, goi ham doublecome d/v tung phan tu

new\_incom = list(map(doubleincome, income))

print(income)

print(new\_incom)

### Dictionary

classmates = {'Tony':'Cao cao', 'Susan':'Hoc gioi', 'Lucy':'Luoi hoc', 'Mary':

'Beo nhut lop'}

print (classmates['Susan'])

for k, v in classmates.items():

print (k + v)

Sử dụng BeautifulSoup4 để lấy phần tử trong các thẻ lồng nhau

Maybe I'm guessing what you are trying to do is first looking in a specific div tag and the search all p tags in it and count them or do whatever you want. For example:

soup = bs4.BeautifulSoup(content, 'html.parser')

# This will get the div

div\_container = soup.find('div', class\_='some\_class')

# Then search in that div\_container for all p tags with class "hello"

for ptag in div\_container.find\_all('p', class\_='hello'):

# prints the p tag content

print(ptag.text)