```
print("20 days are " + str(50) + " minutes")
print(f"20 days are {20 * 24 * 60} minutes")
```

Usando textos e números em python

Variaveis

```
calculation_to_units = 24
name_of_unit = "hour"

print(f"35 days are {35 * calculation_to_units} {name_of_unit}")
print(f"50 days are {50 * calculation_to_units} {name_of_unit}")
print(f"110 days are {110 * calculation_to_units} {name_of_unit}")
print(f"20 days are {20 * calculation_to_units} {name_of_unit}")
```

Funcoes

```
calculation_to_units = 24
name_of_unit = "hours"

def days_to_units(num_of_days, custom_message):
   print(f"{num_of_days} days are {num_of_days * calculation_to_units}
{name_of_unit}")
   print(custom_message)

days_to_units(20, "Awesome!")
days_to_units(35, "Great")
```

Variable scopes in functions - que somente existe onde foi criada - global and local

```
calculation_to_units = 24
name_of_unit = "hours" - VARIAVEIS GLOBAIS

def days_to_units(num_of_days, custom_message):
   print(f"{num_of_days} days are {num_of_days * calculation_to_units}
{name_of_unit}")
   print(custom_message)

def scope_check(num_of_days): VARIAVEIS LOCAIS
   my_var = "variable inside function"
   print(name_of_unit)
   print(num_of_days)
   print(my_var)
scope_check(20)
```

User input

```
calculation_to_units = 24
name_of_unit = "hours"


def days_to_units(num_of_days):
   print(f"{num_of_days} days are {num_of_days * calculation_to_units}
{name_of_unit}")


input("Hey user, enter a number of days and I will convert it to
hours!\n")
```

Para trazer além da pergunta precisa ser especifico usando user_input:

```
calculation_to_units = 24
name_of_unit = "hours"


def days_to_units(num_of_days):
   print(f"{num_of_days} days are {num_of_days * calculation_to_units}
{name_of_unit}")


user_input = input("Hey user, enter a number of days and I will convert it
to hours!\n")
print(user_input)
```

Functions with return values

```
calculation_to_units = 24
name_of_unit = "hours"


def days_to_units(num_of_days):
   return f"{num_of_days} days are {num_of_days * calculation_to_units}
{name_of_unit}"

my_var = days_to_units(20)
print(my_var)
```

```
calculation_to_units = 24
name_of_unit = "hours"


def days_to_units(num_of_days):
   return f"{num_of_days} days are {num_of_days * calculation_to_units}
{name_of_unit}"

user_input = input("Hey user, enter a number of days and I will convert it
to hours!\n")
user_input_number = int(user_input)

calculated_value = days_to_units(user_input_number)
print(calculated_value)
```

Conditionals (if/else) & Boolean Data Type

```
calculation_to_units = 24
name_of_unit = "hours"

def days_to_units(num_of_days):
   condition_check = num_of_days > 0
   print(type(condition_check))

   if num_of_days > 0:
       return f"{num_of_days} days are {num_of_days *
calculation_to_units} {name_of_unit}"
   else:
       return "you entered a negative value, so no conversion for you!"

user_input = input("Hey user, enter a number of days and I will convert it
to hours!\n")
user_input_number = int(user_input) -AQUI ELE MOSTRA COMO RESULTADO <class
'bool'>

calculated_value = days_to_units(int(user_input))
print(calculated_value)
```

types

```
calculation_to_units = 24
```

```
name_of_unit = "hours"

def days_to_units(num_of_days):
   condition_check = num_of_days > 0
   print(type(condition_check))

   if num_of_days > 0:
       return f"{num_of_days} days are {num_of_days *
calculation_to_units} {name_of_unit}"
   else:
       return "you entered a negative value, so no conversion for you!"

print(type("this should br a string"))
```

types

```
calculation_to_units = 24
name_of_unit = "hours"


def days_to_units(num_of_days):
   condition_check = num_of_days > 0
   print(type(condition_check))

   if num_of_days > 0:
       return f"{num_of_days} days are {num_of_days *
calculation_to_units} {name_of_unit}"
   else:
       return "you entered a negative value, so no conversion for you!"

print(type(30.99))
```

conditions

```
calculation_to_units = 24
name_of_unit = "hours"


def days_to_units(num_of_days):
   if num_of_days > 0:
       return f"{num_of_days} days are {num_of_days *
calculation_to_units} {name_of_unit}"
```

```
    elif num_of_days == 0:
        return "you entered a 0, please try again"
    else:
        return "you entered a negative value, so no conversion for you!"

user_input = input("Hey user, enter a number of days and I will convert it
to hours!\n")
user_input_number = int(user_input)

calculated_value = days_to_units(int(user_input))
print(calculated_value)
```

More user input validation

```
calculation_to_units = 24
name_of_unit = "hours"


def days_to_units(num_of_days):
    if num_of_days > 0:
        return f"{num_of_days} days are {num_of_days *
calculation_to_units} {name_of_unit}"
    elif num_of_days == 0:
        return "you entered a 0, please try again"

user_input = input("Hey user, enter a number of days and I will convert it
to hours!\n")


if user_input.isdigit(): -VALIDATION USE IF AND ELSE WHAT CAN MAKE TAKE
OUT THE BEFORE ELSE.
    user_input_number = int(user_input)
    calculated_value = days_to_units(int(user_input))
    print(calculated_value)
else:
    print(("your input isn't a valid number. Don't ruin my program!"))
```

Clean up our code

```
calculation_to_units = 24
```

```
name_of_unit = "hours"


def days_to_units(num_of_days):
   if num_of_days > 0:
       return f"{num_of_days} days are {num_of_days *
calculation_to_units} {name_of_unit}"
   elif num_of_days == 0:
       return "you entered a 0, please try again"

def validate_and_execute():
   if user_input.isdigit():
       user_input_number = int(user_input)
       calculated_value = days_to_units(int(user_input))
       print(calculated_value)
   else:
       print(("your input isn't a valid number. Don't ruin my program!"))

user_input = input("Hey user, enter a number of days and I will convert it
to hours!\n")
validate_and_execute()
```

Nested If… else

```
calculation_to_units = 24
name_of_unit = "hours"


def days_to_units(num_of_days):
   return f"{num_of_days} days are {num_of_days * calculation_to_units}
{name_of_unit}"


def validate_and_execute():
   if user_input.isdigit():
       user_input_number = int(user_input)
       if user_input_number > 0:
           calculated_value = days_to_units(int(user_input))
           print(calculated_value)
       elif user_input_number == 0:
```

```
            print("you entered a 0, please try again")
    else:
        print(("your input isn't a valid number. Don't ruin my program!"))

user_input = input("Hey user, enter a number of days and I will convert it
to hours!\n")
validate_and_execute()
```

Error Handling with try/except

```
calculation_to_units = 24
name_of_unit = "hours"


def days_to_units(num_of_days):
    return f"{num_of_days} days are {num_of_days * calculation_to_units}
{name_of_unit}"


def validate_and_execute():
    try:

        user_input_number = int(user_input)
        if user_input_number > 0:
            calculated_value = days_to_units(int(user_input))
            print(calculated_value)
        elif user_input_number == 0:
            print("you entered a 0, please try again")
        else:
            print("you entered a negative number, no conversion for you!")

    except ValueError:
        print(("your input isn't a valid number. Don't ruin my program!"))

user_input = input("Hey user, enter a number of days and I will convert it
to hours!\n")
validate_and_execute()
```

While loops

```
calculation_to_units = 24
name_of_unit = "hours"

def days_to_units(num_of_days):
   return f"{num_of_days} days are {num_of_days * calculation_to_units}
{name_of_unit}"

def validate_and_execute():

   try:
       user_input_number = int(user_input)
       if user_input_number > 0:
           calculated_value = days_to_units(int(user_input))
           print(calculated_value)
       elif user_input_number == 0:
           print("you entered a 0, please try again")
       else:
           print("you entered a negative number, no conversion for you!")

   except ValueError:
       print(("your input isn't a valid number. Don't ruin my program!"))

while True:
   user_input = input("Hey user, enter a number of days and I will convert
it to hours!\n")
   validate_and_execute()
```

Let user exit the program

```
calculation_to_units = 24
name_of_unit = "hours"

def days_to_units(num_of_days):
   return f"{num_of_days} days are {num_of_days * calculation_to_units}
{name_of_unit}"

def validate_and_execute():
```

```
    try:
        user_input_number = int(user_input)
        if user_input_number > 0:
            calculated_value = days_to_units(int(user_input))
            print(calculated_value)
        elif user_input_number == 0:
            print("you entered a 0, please try again")
        else:
            print("you entered a negative number, no conversion for you!")

    except ValueError:
        print(("your input isn't a valid number. Don't ruin my program!"))

user_input = ""

while user_input != "exit":
    user_input = input("Hey user, enter a number of days and I will convert
it to hours!\n")
    validate_and_execute()
```

Lists & For Loops

Split> transforma tudo em lista, podendo adicionar mais de um elemento no terminal para verificação e ele traz mais de um resultado em string e o try transforme tudo depois em integer: Além disso tem o exemplo de como ver o que acontece dentro do split e que type ele traz.

```
calculation_to_units = 24
name_of_unit = "hours"

def days_to_units(num_of_days):
    return f"{num_of_days} days are {num_of_days * calculation_to_units}
{name_of_unit}"

def validate_and_execute():
    try:
        user_input_number = int(num_of_days_element)

        if user_input_number > 0:
            calculated_value = days_to_units(user_input_number)
            print(calculated_value)
        elif user_input_number == 0:
            print("you entered a 0, please try again")
```

```
        else:
            print("you entered a negative number, no conversion for you!")

    except ValueError:
        print(("your input isn't a valid number. Don't ruin my program!"))

user_input = ""
while user_input != "exit":
    user_input = input("Hey user, enter a number of days as a comma
separated list and I will convert it to hours!\n")
    print(type(user_input.split(", ")))
    print(user_input.split(", "))

    for num_of_days_element in user_input.split(", "):
        validate_and_execute()
```

Basic List Operations
Para fazer uma lista aparecer no terminal usando array e printando eles.
Append: adiciona o valor na lista e printando o valor ele também aparecerá mas se colocar outro que não existe ele dá erro.
In new document lists.py I put:

```
my_list = ["January", "February", "March"]
print(my_list[2])
my_list.append("April")
print(my_list[4])
```

Comments
and comment for line or
“”” comment all text and after you close with 3 aspas “””
Or ctrl+/

```
calculation_to_units = 24
name_of_unit = "hours"

def days_to_units(num_of_days):
    return f"{num_of_days} days are {num_of_days * calculation_to_units}
{name_of_unit}"

def validate_and_execute():
    try:
        user_input_number = int(num_of_days_element)
```

```
        # we want to do conversion only for positive integers
        if user_input_number > 0:
            calculated_value = days_to_units(user_input_number)
            print(calculated_value)
        elif user_input_number == 0:
            print("you entered a 0, please try again")
        else:
            print("you entered a negative number, no conversion for you!")

    except ValueError:
        print(("your input isn't a valid number. Don't ruin my program!"))

user_input = ""
while user_input != "exit":
    user_input = input("Hey user, enter a number of days as a comma
separated list and I will convert it to hours!\n")
    print(type(user_input.split(", ")))
    print(user_input.split(", "))

    for num_of_days_element in user_input.split(", "):
        validate_and_execute()
```

Sets

Its another build-in data type of Python, as with Lists , used to store multiple items of data, does not allow duplicate values

```
calculation_to_units = 24
name_of_unit = "hours"

def days_to_units(num_of_days):
    return f"{num_of_days} days are {num_of_days * calculation_to_units}
{name_of_unit}"

def validate_and_execute():
    try:
        user_input_number = int(num_of_days_element)

        # we want to do conversion only for positive integers
        if user_input_number > 0:
            calculated_value = days_to_units(user_input_number)
            print(calculated_value)
```

```
        elif user_input_number == 0:
            print("you entered a 0, please try again")
        else:
            print("you entered a negative number, no conversion for you!")

    except ValueError:
        print(("your input isn't a valid number. Don't ruin my program!"))

user_input = ""
while user_input != "exit":
    user_input = input("Hey user, enter a number of days as a comma
separated list and I will convert it to hours!\n")

    list_of_days = user_input.split(", ")

    print(list_of_days)
    print(set(list_of_days))
    print(type(list_of_days))
    print(type(set(list_of_days)))

    for num_of_days_element in set(list_of_days):
        validate_and_execute()
```

TERMINAL

```
Hey user, enter a number of days as a comma separated list and I will
convert it to hours!
10, 45, 30, 10
['10', '45', '30', '10']
{'45', '10', '30'}
<class 'list'>
<class 'set'>
45 days are 1080 hours
10 days are 240 hours
30 days are 720 hours
Hey user, enter a number of days as a comma separated list and I will
convert it to hours!
```

Basic set operations & syntax

And a new document sets.py this its different of lists.py because only work sets with a loop and the result it is not an order.

```
my_set = {"January", "February", "March"}
for element in my_set:
   print(element)

my_set.add("April")
print(my_set)

my_set.remove("January")
print(my_set)

my_list = ["January", "February", "March", "January"]
my_list.remove("January")
print(my_list)



                                TERMINAL


February
March
January

{'April', 'February', 'March', 'January'}

{'April', 'February', 'March'}

['February', 'March', 'January']
```

Build-in Functions

Standalone Build-in Functions = print() to the standard output device; input() ask users for input; returns a new set; int() converts value into a integer number

```
print("some text")
input("enter value")
set([1, 3, 5])
int("20")

"2, 3".split()
"text".isdigit()
[1, 3, 4].count()
```

Dictionary data type
Como criar:

```
calculation_to_units = 24
name_of_unit = "hours"

def days_to_units(num_of_days):
   return f"{num_of_days} days are {num_of_days * calculation_to_units}
{name_of_unit}"

def validate_and_execute():
   try:
       user_input_number = int(days_and_unit_dictionary["days"])

       # we want to do conversion only for positive integers
       if user_input_number > 0:
           calculated_value = days_to_units(user_input_number)
           print(calculated_value)
       elif user_input_number == 0:
           print("you entered a 0, please try again")
       else:
           print("you entered a negative number, no conversion for you!")

   except ValueError:
       print(("your input isn't a valid number. Don't ruin my program!"))

# user_input = ""
# while user_input != "exit":
#     user_input = input("Hey user, enter a number of days and conversion
unit!\n")
#     days_and_unit = user_input.split(":")
#     print(days_and_unit)

#     days_and_unit_dictionary = {"days": days_and_unit[0], "unit":
days_and_unit[1]}
#     print(days_and_unit_dictionary)
#     validate_and_execute()

my_list = ["20", "30", "100"]
print(my_list[2])
```

```
my_dictionary = {"days": 20, "unit" : "hours", "message": "all good"}
print(my_dictionary["days"])
print(my_dictionary["unit"])
print(my_dictionary["message"])

                                TERMINAL

100

20
hours
all good
```

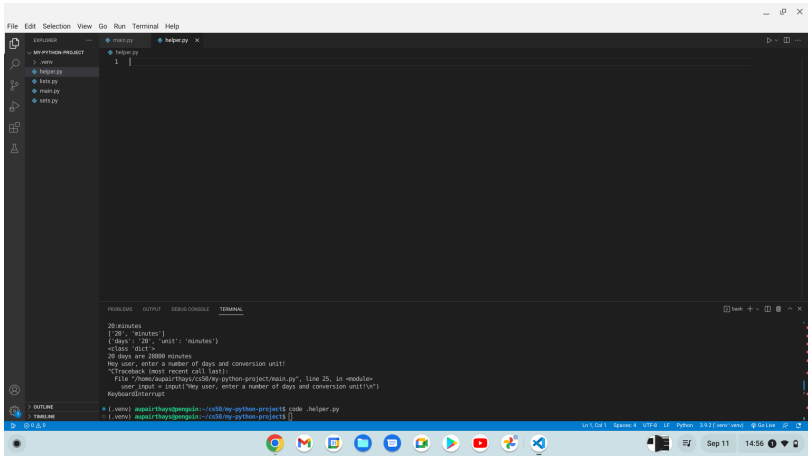
Data types

```
message = "enter some value"
days = 20
price = 9.99
valid_number = True
exit_input = False
list_of_days = [20, 40, 20, 100]
list_of_months = ["January", "February", "June"]
set_of_days = {20, 45, 100}
days_and_unit = {"days": 10, "unit": "hours"}
```

```
def days_to_units(num_of_days, conversion_unit):
    if conversion_unit == "hours":
        return f"{num_of_days} days are {num_of_days * 24} hours"
    elif conversion_unit == "minutes":
        return f"{num_of_days} days are {num_of_days * 24 * 60} minutes"
    else:
        return "unsupported unit"

def validate_and_execute():
    try:
        user_input_number = int(days_and_unit_dictionary["days"])
        if user_input_number > 0:
            calculated_value = days_to_units(user_input_number,
days_and_unit_dictionary["unit"])
            print(calculated_value)
```

```
        elif user_input_number == 0:
            print("you entered a 0, please try again")
        else:
            print("you entered a negative number, no conversion for you!")

    except ValueError:
        print(("your input isn't a valid number. Don't ruin my program!"))

user_input = ""
while user_input != "exit":
    user_input = input("Hey user, enter a number of days and conversion
unit!\n")
    days_and_unit = user_input.split(":")
    print(days_and_unit)

    days_and_unit_dictionary = {"days": days_and_unit[0], "unit":
days_and_unit[1]}
    print(days_and_unit_dictionary)
    print(type(days_and_unit_dictionary))
    validate_and_execute()
```

Modules:
Logically organize your Python code
Module should contain related code
Is just a .py file



Create a Module & import statement

Main.py

```
import helper

user_input = ""
while user_input != "exit":
   user_input = input("Hey user, enter a number of days and conversion
unit!\n")
   days_and_unit = user_input.split(":")
   days_and_unit_dictionary = {"days": days_and_unit[0], "unit":
days_and_unit[1]}
   helper.validate_and_execute(days_and_unit_dictionary)
```

Helper.py

```
def days_to_units(num_of_days, conversion_unit):
   if conversion_unit == "hours":
       return f"{num_of_days} days are {num_of_days * 24} hours"
   elif conversion_unit == "minutes":
       return f"{num_of_days} days are {num_of_days * 24 * 60} minutes"
   else:
       return "unsupported unit"

def validate_and_execute(days_and_unit_dictionary):
   try:
       user_input_number = int(days_and_unit_dictionary["days"])
       if user_input_number > 0:
           calculated_value = days_to_units(user_input_number,
days_and_unit_dictionary["unit"])
           print(calculated_value)
       elif user_input_number == 0:
           print("you entered a 0, please try again")
       else:
           print("you entered a negative number, no conversion for you!")

   except ValueError:
       print(("your input isn't a valid number. Don't ruin my program!"))
```

Or

Main.py

```
from helper import validate_and_execute

user_input = ""
while user_input != "exit":
   user_input = input("Hey user, enter a number of days and conversion
unit!\n")
   days_and_unit = user_input.split(":")
   days_and_unit_dictionary = {"days": days_and_unit[0], "unit":
days_and_unit[1]}
   validate_and_execute(days_and_unit_dictionary)
```

Helper.py

```
def days_to_units(num_of_days, conversion_unit):
   if conversion_unit == "hours":
       return f"{num_of_days} days are {num_of_days * 24} hours"
   elif conversion_unit == "minutes":
       return f"{num_of_days} days are {num_of_days * 24 * 60} minutes"
   else:
       return "unsupported unit"

def validate_and_execute(days_and_unit_dictionary):
   try:
       user_input_number = int(days_and_unit_dictionary["days"])
       if user_input_number > 0:
           calculated_value = days_to_units(user_input_number,
days_and_unit_dictionary["unit"])
           print(calculated_value)
       elif user_input_number == 0:
           print("you entered a 0, please try again")
       else:
           print("you entered a negative number, no conversion for you!")

   except ValueError:
       print(("your input isn't a valid number. Don't ruin my program!"))
```

Main.py

```
from helper import validate_and_execute, user_input_message
```

```
user_input = ""
while user_input != "exit":
   user_input = input(user_input_message)
   days_and_unit = user_input.split(":")
   days_and_unit_dictionary = {"days": days_and_unit[0], "unit":
days_and_unit[1]}
   validate_and_execute(days_and_unit_dictionary)
```

Helper.py

```
def days_to_units(num_of_days, conversion_unit):
   if conversion_unit == "hours":
       return f"{num_of_days} days are {num_of_days * 24} hours"
   elif conversion_unit == "minutes":
       return f"{num_of_days} days are {num_of_days * 24 * 60} minutes"
   else:
       return "unsupported unit"

def validate_and_execute(days_and_unit_dictionary):
   try:
       user_input_number = int(days_and_unit_dictionary["days"])
       if user_input_number > 0:
           calculated_value = days_to_units(user_input_number,
days_and_unit_dictionary["unit"])
           print(calculated_value)
       elif user_input_number == 0:
           print("you entered a 0, please try again")
       else:
           print("you entered a negative number, no conversion for you!")

   except ValueError:
       print(("your input isn't a valid number. Don't ruin my program!"))

user_input_message = "Hey user, enter a number of days and conversion
unit!\n"
```

Importa tudo

```
from helper import *
```

Other way

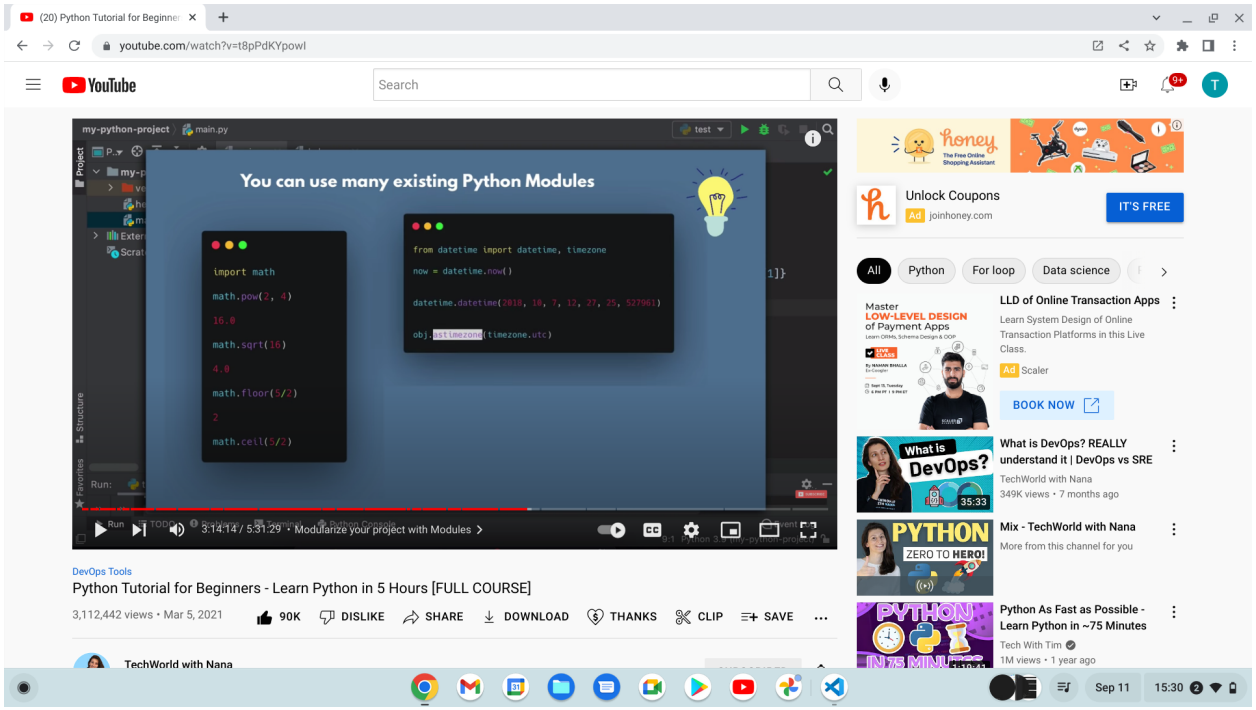
```
import helper

user_input = ""
while user_input != "exit":
   user_input = input(helper.user_input_message)
   days_and_unit = user_input.split(":")
   days_and_unit_dictionary = {"days": days_and_unit[0], "unit":
days_and_unit[1]}
   helper.validate_and_execute(days_and_unit_dictionary)
```

```
def days_to_units(num_of_days, conversion_unit):
   if conversion_unit == "hours":
       return f"{num_of_days} days are {num_of_days * 24} hours"
   elif conversion_unit == "minutes":
       return f"{num_of_days} days are {num_of_days * 24 * 60} minutes"
   else:
       return "unsupported unit"

def validate_and_execute(days_and_unit_dictionary):
   try:
       user_input_number = int(days_and_unit_dictionary["days"])
       if user_input_number > 0:
           calculated_value = days_to_units(user_input_number,
days_and_unit_dictionary["unit"])
           print(calculated_value)
       elif user_input_number == 0:
           print("you entered a 0, please try again")
       else:
           print("you entered a negative number, no conversion for you!")

   except ValueError:
       print(("your input isn't a valid number. Don't ruin my program!"))

user_input_message = "Hey user, enter a number of days and conversion
unit!\n"
```

To names to big

```
import helper as h
```

Built-in python modules



Built-in Module examples

```
import os

print(os.name)

                                TERMINAL

posix
```

```
import logging

logger = logging.getLogger("MAIN")
logger.error("ERROR HAPPENED IN THE APP")

                                TERMINAL

ERROR HAPPENED IN THE APP
```

PROJECT
COUNTDOWN APP
EXERCISE

```
import datetime

user_input = input("enter your goal with a deadline separated by colon\n")
```

```
input_list = user_input.split(":")

goal = input_list[0]
deadline = input_list[1]
print(input_list)
```

enter your goal with a deadline separated by colon
learn python:10.02.2021
['learn python', '10.02.2021']

```
import datetime

user_input = input("enter your goal with a deadline separated by colon\n")
input_list = user_input.split(":")

goal = input_list[0]
deadline = input_list[1]

print(datetime.datetime.strptime(deadline, "%d.%m.%Y"))

print(type(datetime.datetime.strptime(deadline, "%d.%m.%Y")))
```

TERMINAL

```
learn python:10.02.2021
2021-02-10 00:00:00
<class 'datetime.datetime'>
```

```
import datetime

user_input = input("enter your goal with a deadline separated by colon\n")
input_list = user_input.split(":")

goal = input_list[0]
deadline = input_list[1]

deadline_date = datetime.datetime.strptime(deadline, "%d.%m.%Y")
```

```
# calculate how many days from now till deadline

print(datetime.datetime.today())
```

TERMINAL

```
enter your goal with a deadline separated by colon
learn python:10.02.2021
2022-09-11 15:56:39.251394
```

```
import datetime

user_input = input("enter your goal with a deadline separated by colon\n")
input_list = user_input.split(":")

goal = input_list[0]
deadline = input_list[1]

deadline_date = datetime.datetime.strptime(deadline, "%d.%m.%Y")
today_date = datetime.datetime.today()
# calculate how many days from now till deadline

print(deadline_date - today_date)
```

TERMINAL

```
enter your goal with a deadline separated by colon
learn python:10.02.2023
151 days, 7:58:59.346588
```

```
import datetime

user_input = input("enter your goal with a deadline separated by colon\n")
input_list = user_input.split(":")

goal = input_list[0]
deadline = input_list[1]

deadline_date = datetime.datetime.strptime(deadline, "%d.%m.%Y")
today_date = datetime.datetime.today()
time_till = deadline_date - today_date
```

```
print(f"Dear user! Time remaining for your goal: {goal} is {time_till}")
```

TERMINAL

```
enter your goal with a deadline separated by colon
learn python:10.02.2023
Dear user! Time remaining for your goal: learn python is 151 days,
7:53:42.530854
```

```
import datetime

user_input = input("enter your goal with a deadline separated by colon\n")
input_list = user_input.split(":")

goal = input_list[0]
deadline = input_list[1]

deadline_date = datetime.datetime.strptime(deadline, "%d.%m.%Y")
today_date = datetime.datetime.today()
time_till = deadline_date - today_date

print(f"Dear user! Time remaining for your goal: {goal} is
{time_till.days} days")
```

TERMINAL

```
enter your goal with a deadline separated by colon
learn python:10.02.2023
Dear user! Time remaining for your goal: learn python is 151 days,
7:53:42.530854
```

```
import datetime

user_input = input("enter your goal with a deadline separated by colon\n")
input_list = user_input.split(":")

goal = input_list[0]
deadline = input_list[1]

deadline_date = datetime.datetime.strptime(deadline, "%d.%m.%Y")
```

```
today_date = datetime.datetime.today()
time_till = deadline_date - today_date

print(f"Dear user! Time remaining for your goal: {goal} is
{time_till.total_seconds() / 60 / 60} hours")
```

TERMINAL

```
enter your goal with a deadline separated by colon
learn python:10.02.2023
Dear user! Time remaining for your goal: learn python is 3631.805241601389
hours
```

```
import datetime

user_input = input("enter your goal with a deadline separated by colon\n")
input_list = user_input.split(":")

goal = input_list[0]
deadline = input_list[1]

deadline_date = datetime.datetime.strptime(deadline, "%d.%m.%Y")
today_date = datetime.datetime.today()
time_till = deadline_date - today_date

print(f"Dear user! Time remaining for your goal: {goal} is
{int(time_till.total_seconds() / 60 / 60)} hours")
```

TERMINAL

```
enter your goal with a deadline separated by colon
learn python:10.02.2023
Dear user! Time remaining for your goal: learn python is 3631 hours
```

```
import datetime

user_input = input("enter your goal with a deadline separated by colon\n")
input_list = user_input.split(":")
```

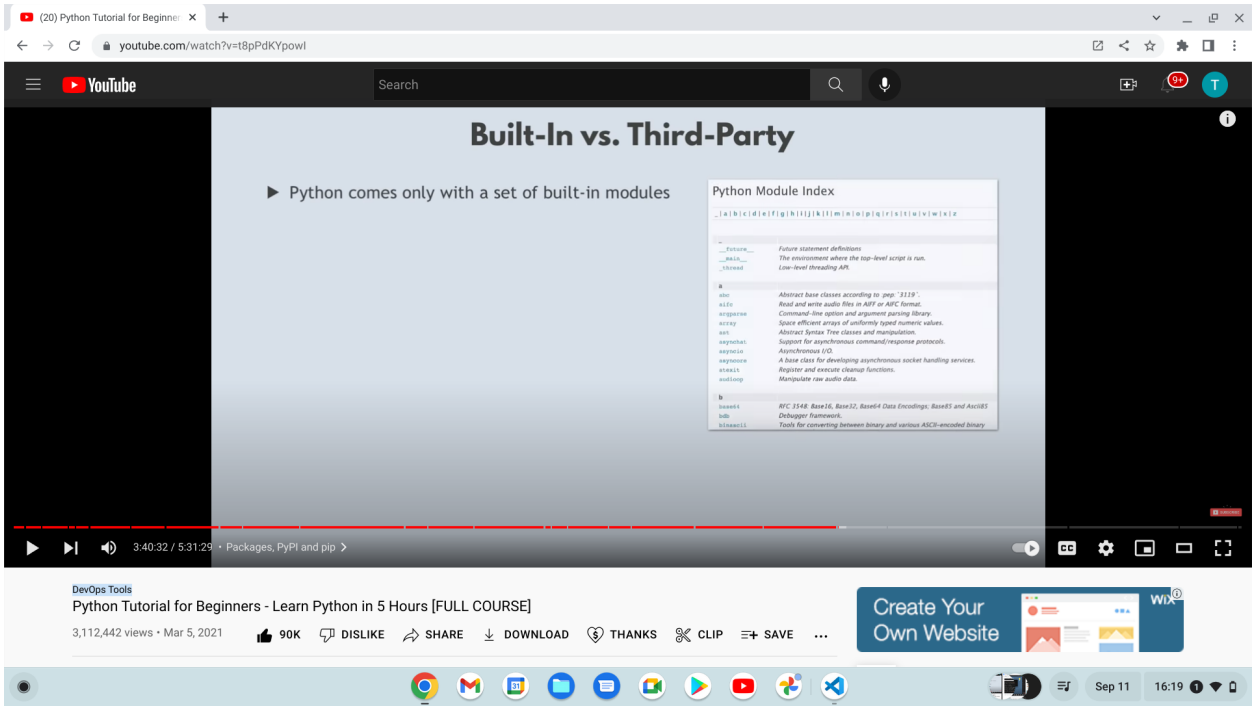
```
goal = input_list[0]
deadline = input_list[1]

deadline_date = datetime.datetime.strptime(deadline, "%d.%m.%Y")
today_date = datetime.datetime.today()
time_till = deadline_date - today_date

hours_till = int(time_till.total_seconds() / 60 / 60)
print(f"Dear user! Time remaining for your goal: {goal} is {hours_till}
hours")
```

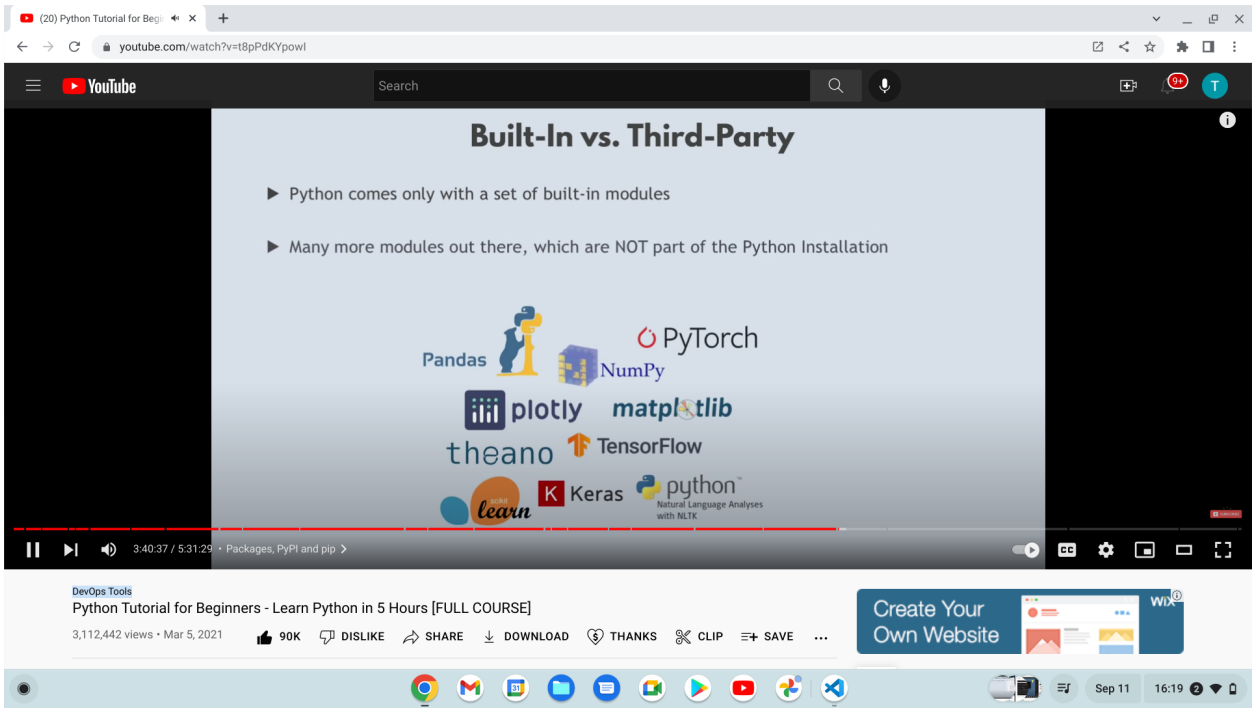
TERMINAL

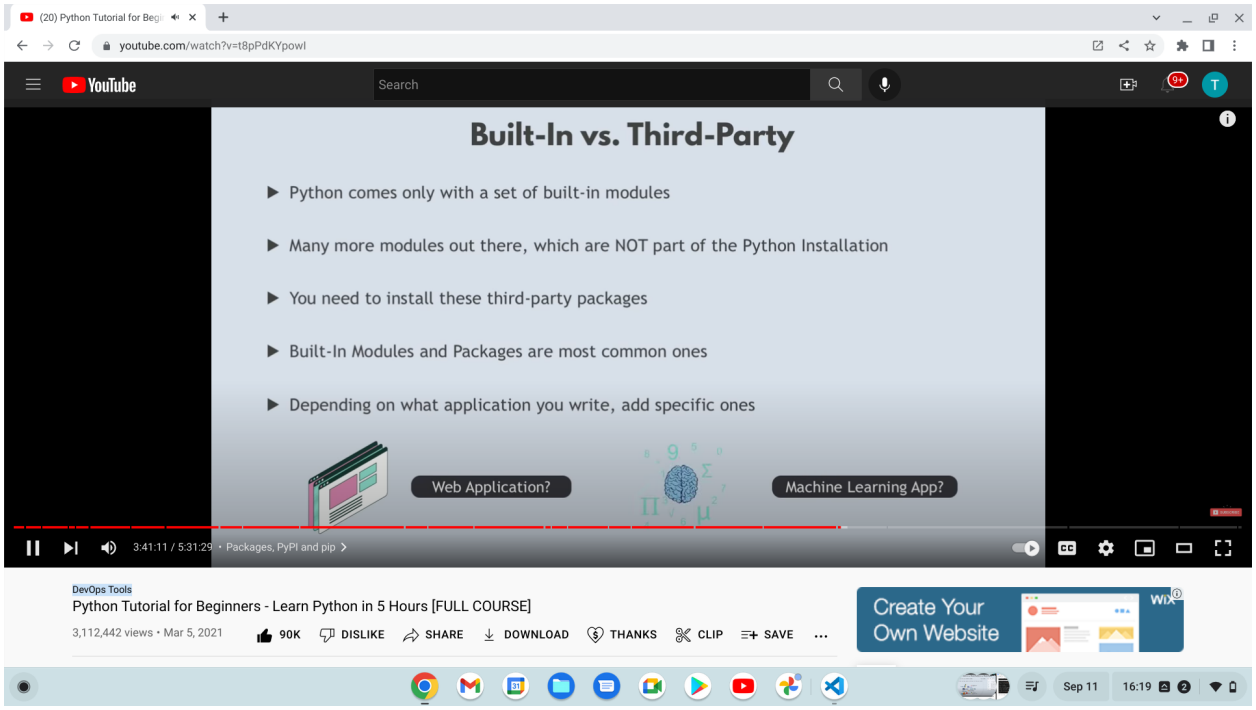
```
enter your goal with a deadline separated by colon
learn python:10.02.2023
Dear user! Time remaining for your goal: learn python is 3631 hours
```

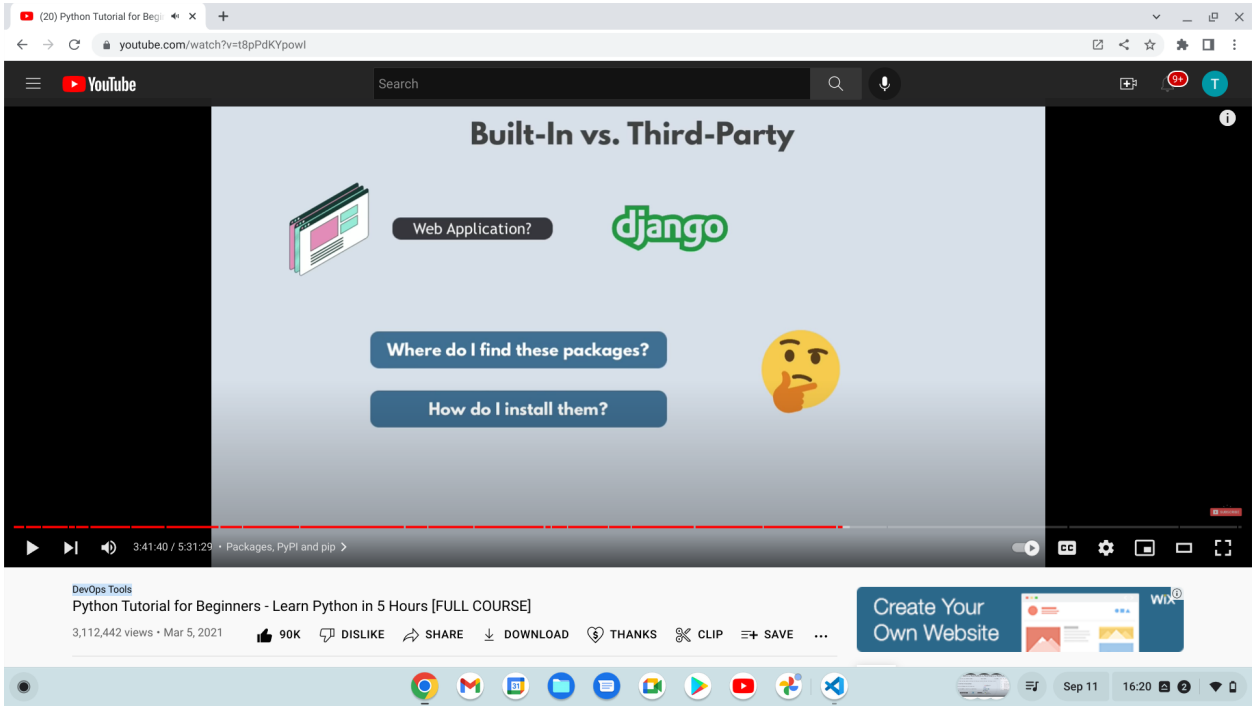
```
from datetime import datetime

user_input = input("enter your goal with a deadline separated by colon\n")
input_list = user_input.split(":")

goal = input_list[0]
deadline = input_list[1]

deadline_date = datetime.strptime(deadline, "%d.%m.%Y")
today_date = datetime.today()
time_till = deadline_date - today_date

hours_till = int(time_till.total_seconds() / 60 / 60)
print(f"Dear user! Time remaining for your goal: {goal} is {hours_till}
hours")
```

Third-party packages


(20) Python Tutorial for Beginner
youtube.com/watch?v=t8pPdKYpowI
YouTube
Search
Built-In vs. Third-Party
Python comes only with a set of built-in modules
Python Module Index
3:40:32 / 5:31:29 • Packages, PyPI and pip
DevOps Tools
Python Tutorial for Beginners - Learn Python in 5 Hours [FULL COURSE]
3,112,442 views • Mar 5, 2021
90K DISLIKE SHARE DOWNLOAD THANKS CLIP SAVE
Create Your Own Website
Sep 11
16:19


(20) Python Tutorial for Begi
youtube.com/watch?v=t8pPdKYpowI
YouTube
Search
Built-In vs. Third-Party
Python comes only with a set of built-in modules
Many more modules out there, which are NOT part of the Python Installation
Pandas
NumPy
PyTorch
plotly
matplotlib
theano
TensorFlow
Keras
python
Natural Language Analyses with NLTK
3:40:37 / 5:31:29 • Packages, PyPI and pip
DevOps Tools
Python Tutorial for Beginners - Learn Python in 5 Hours [FULL COURSE]
3,112,442 views • Mar 5, 2021
90K DISLIKE SHARE DOWNLOAD THANKS CLIP SAVE
Create Your Own Website
Sep 11
16:19

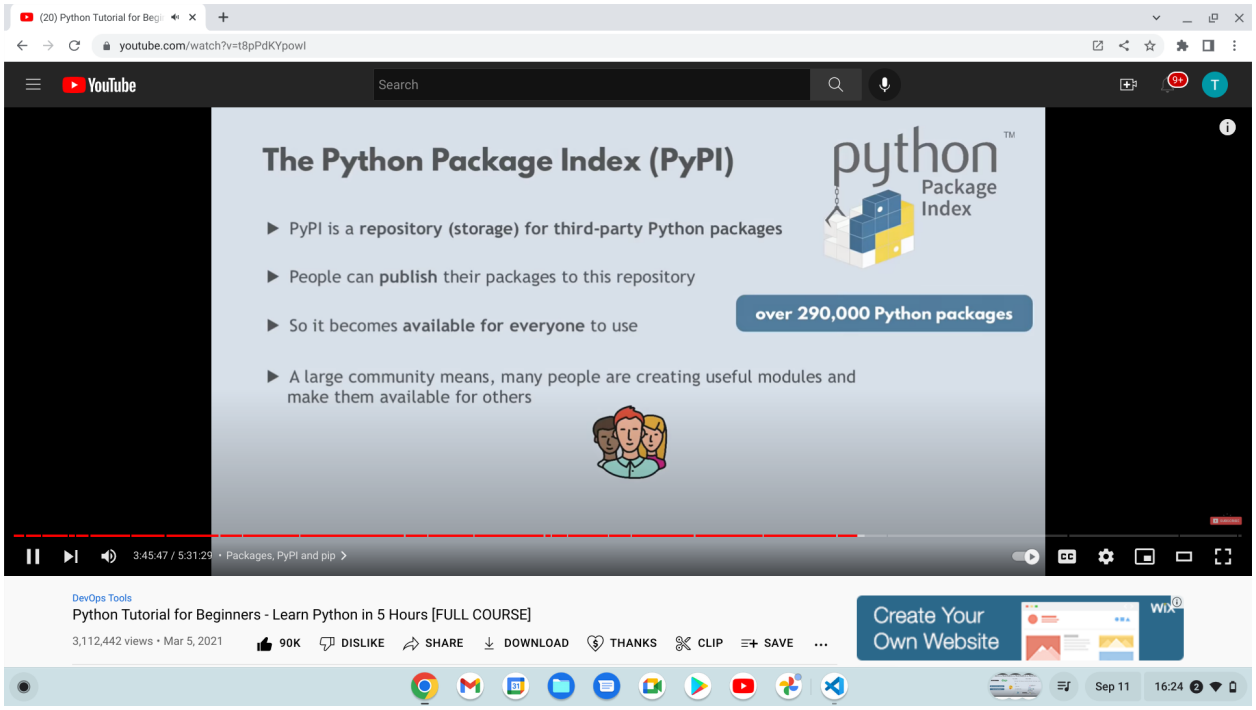




MODULE VS PACKAGE

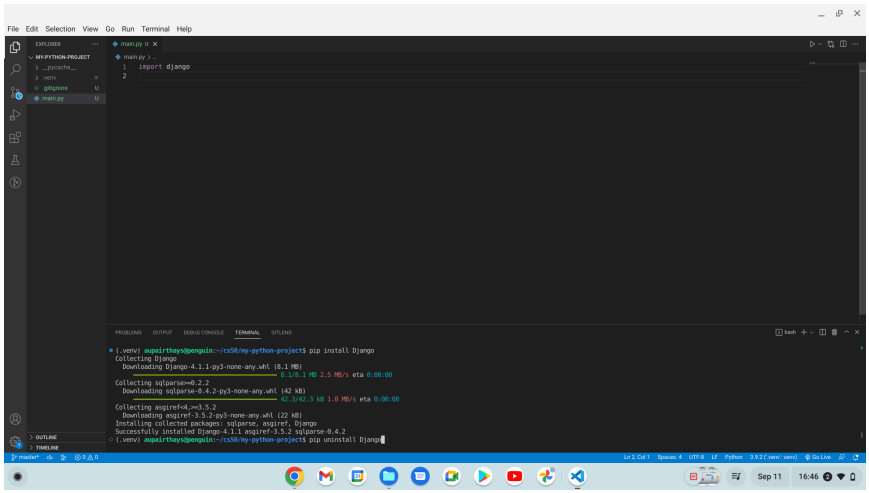
Module any python file is a module
Package is a collection of python modules, must include an_init_.py file, this doc distinguishes a package from a directory.

Para use packages of any web application only look for the site of Python



Pip
No site https://pypi.org/
Pesquise sobre a package que deseja nesse caso ela fala sobre django
Pip is a package manager for Python, used to install packages from the Python Package Index, but also other indexes:
pip install Django
pip uninstall Django

```
Found existing installation: Django 4.1.1
Uninstalling Django-4.1.1:
  Would remove:
    /home/aupairthays/cs50/my-python-project/.venv/bin/django-admin

/home/aupairthays/cs50/my-python-project/.venv/lib/python3.9/site-packages
/Django-4.1.1.dist-info/*

/home/aupairthays/cs50/my-python-project/.venv/lib/python3.9/site-packages
/django/*
Proceed (Y/n)? Y
  Successfully uninstalled Django-4.1.1
```

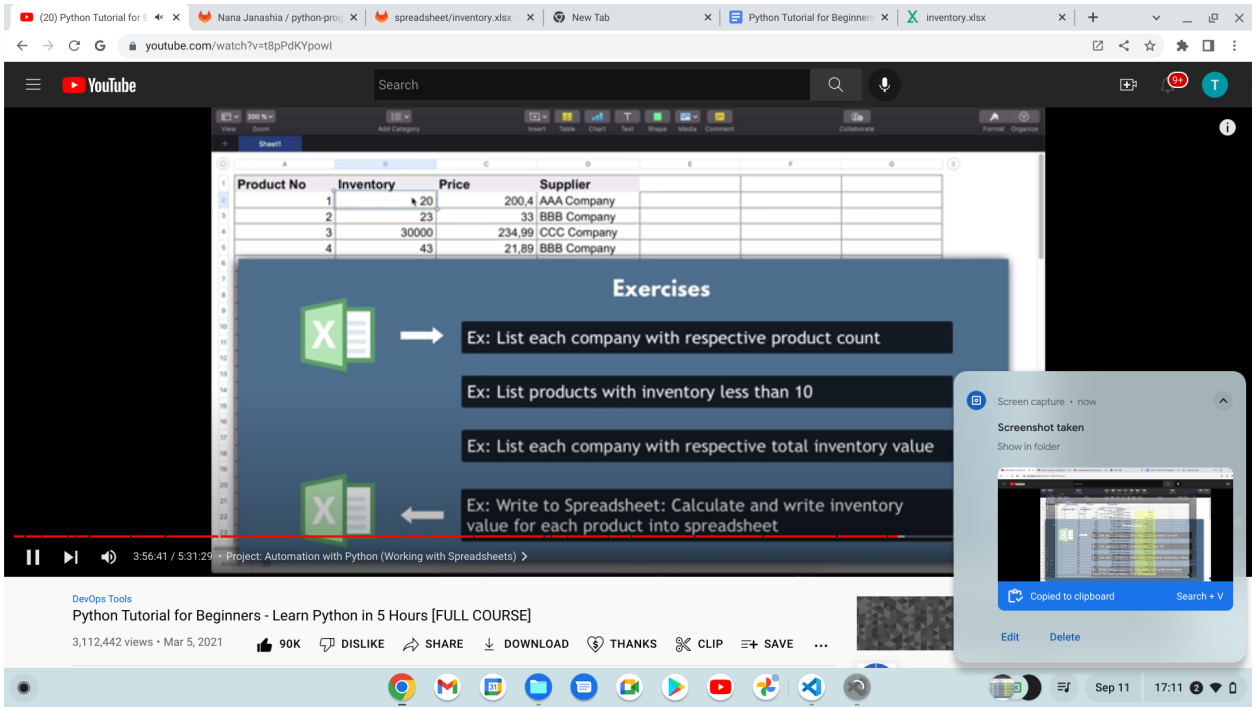


Python Packages Tool Window

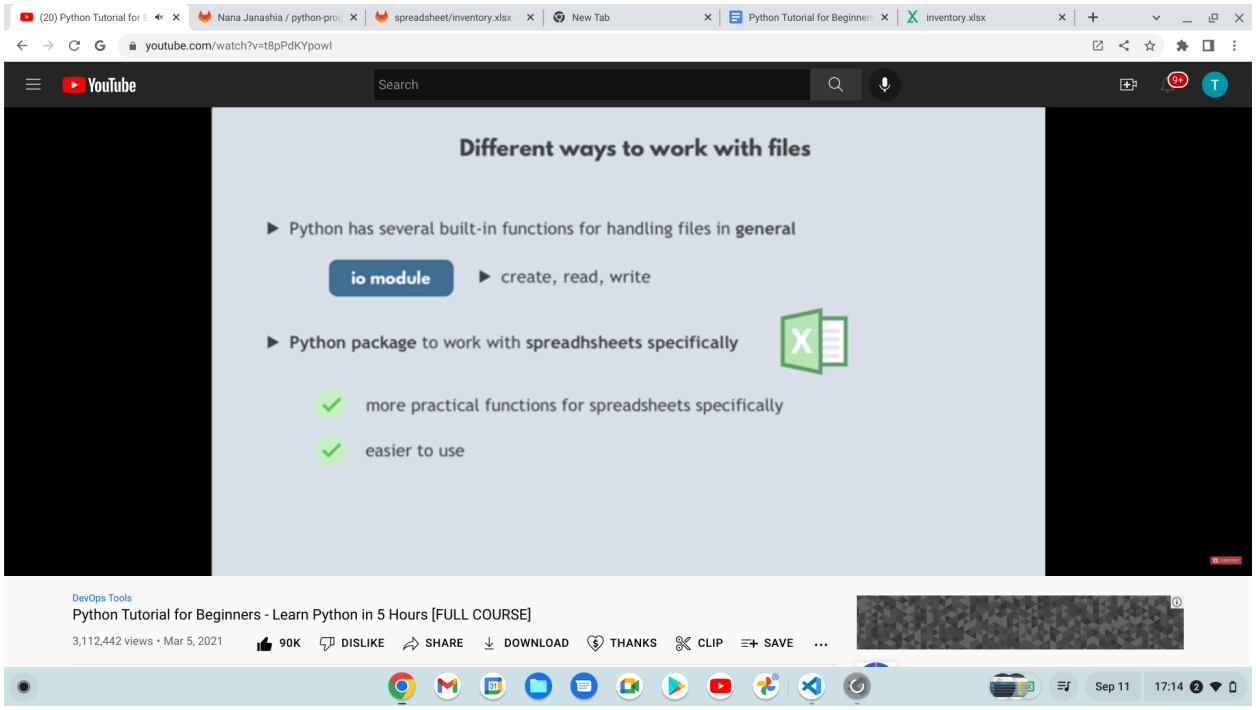
https://www.youtube.com/watch?v=t8pPdKYpowI

https://gitlab.com/nanuchi/python-programming

https://gitlab.com/nanuchi/python-programming/-/tree/master/spreadsheet

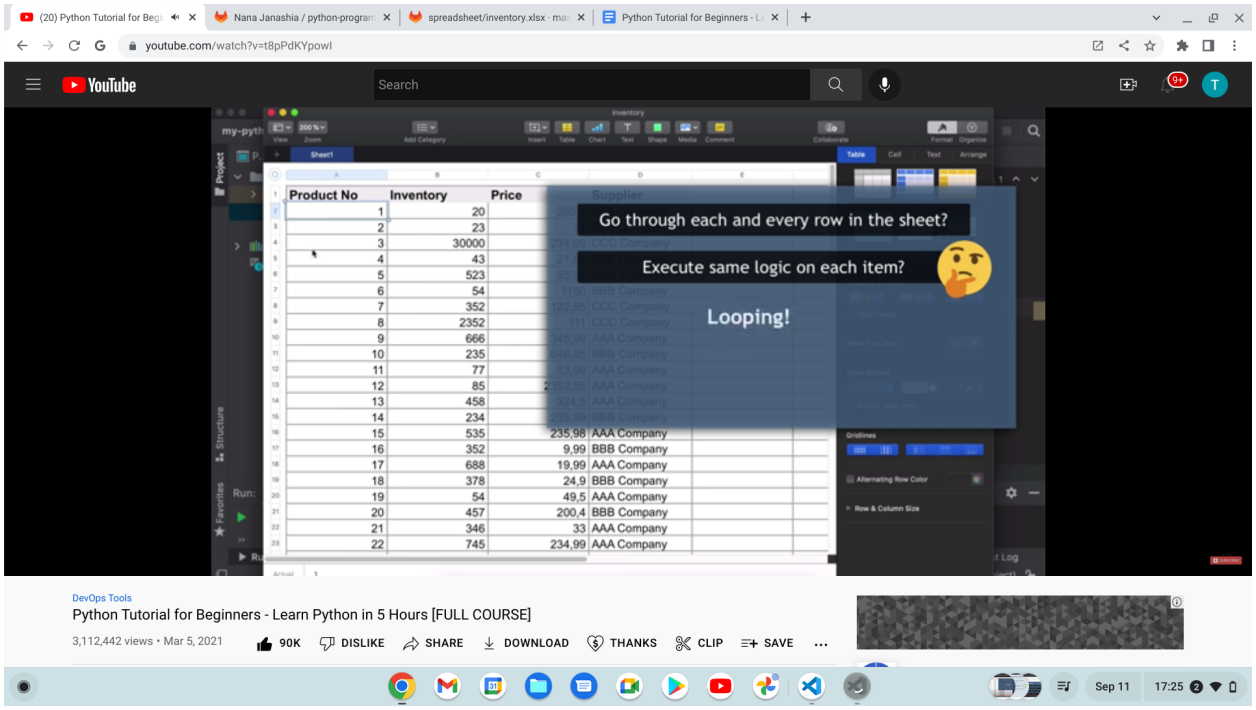


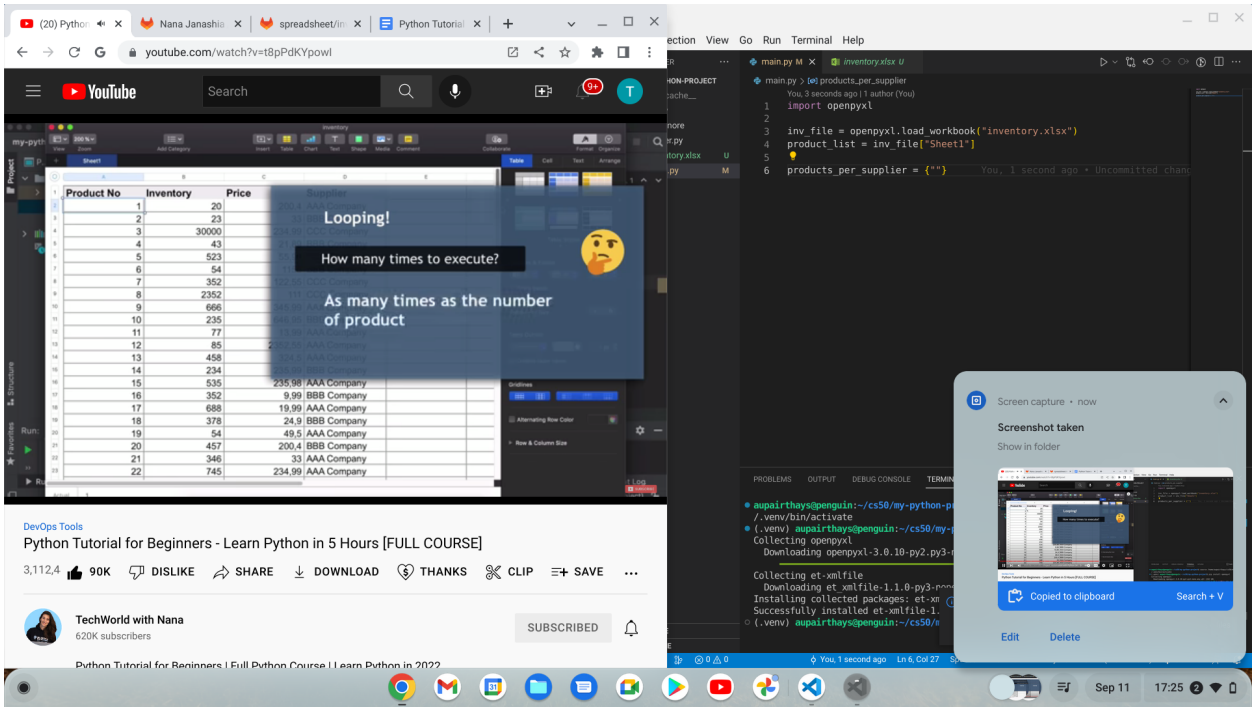
Install package



No site https://pypi.org/
Arquivo openpyxl
pip install openpyxl
After cria o arquivo e abre a planilha

```
import openpyxl

inv_file = openpyxl.load_workbook("inventory.xlsx")
product_list = inv_file["Sheet1"]
```





Primeiro ela printou o nome das colunas

```
import openpyxl


inv_file = openpyxl.load_workbook("inventory.xlsx")
product_list = inv_file["Sheet1"]
```

```
products_per_supplier = {}

print(product_list.max_row)

for product_row in range(2, product_list.max_row + 1):
   supplier_name = product_list.cell(product_row, 4)
   print(supplier_name)

   # if supplier_name in products_per_supplier:
   #     current_num_products = products_per_supplier[supplier_name]
   #     current_per_suppier[supplier_name] = current_num_products + 1
   # else:
   #     print("Adding a new supplier")
   #     products_per_supplier[supplier_name] = 1
# if supplier_name in products_per_supplier:
   #     current_num_products = products_per_supplier[supplier_name]
   #     current_per_suppier[supplier_name] = current_num_products + 1
   # else:
   #     print("Adding a new supplier")
   #     products_per_supplier[supplier_name] = 1


<Cell 'Sheet1'.D7>
<Cell 'Sheet1'.D8>
<Cell 'Sheet1'.D9>
<Cell 'Sheet1'.D10>
```

Ela acrescentou value

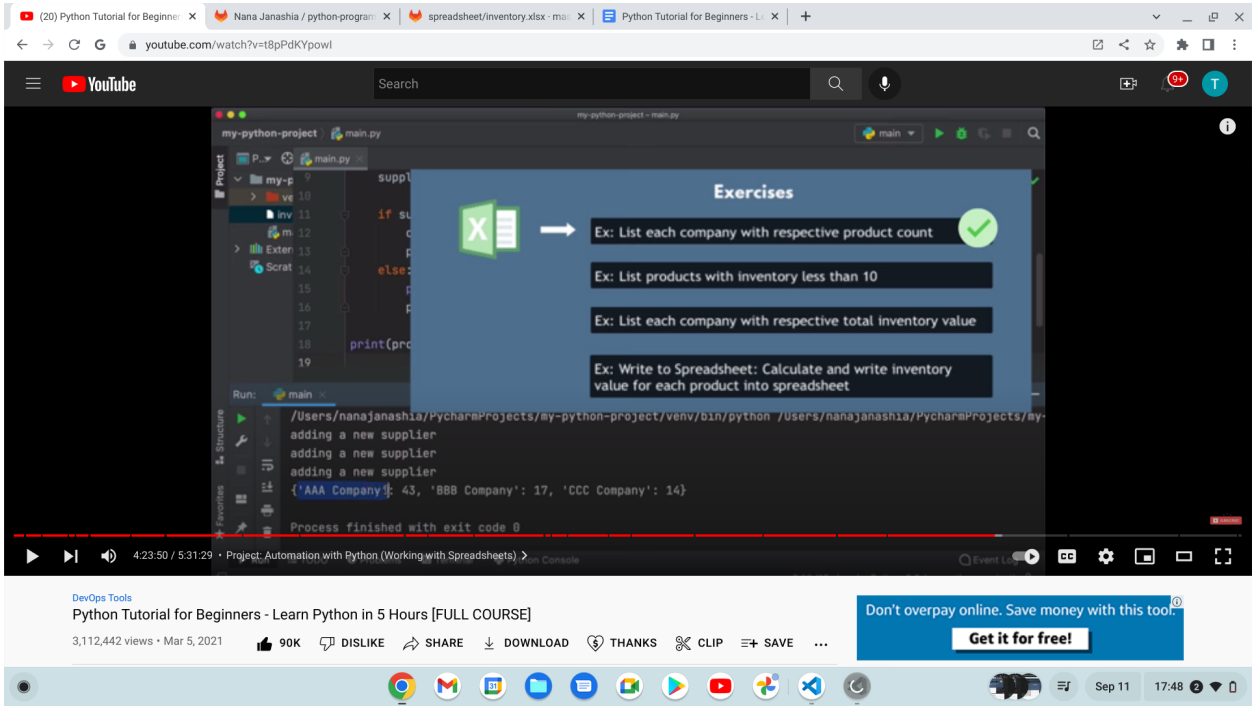
```
import openpyxl

inv_file = openpyxl.load_workbook("inventory.xlsx")
product_list = inv_file["Sheet1"]

products_per_supplier = {}

print(product_list.max_row)

for product_row in range(2, product_list.max_row + 1):
```

```
    supplier_name = product_list.cell(product_row, 4).value
    # print(supplier_name)

    # if supplier_name in products_per_supplier:
    #     current_num_products = products_per_supplier[supplier_name]
    #     current_per_suppier[supplier_name] = current_num_products + 1
    # else:
    #     print("Adding a new supplier")
    #     products_per_supplier[supplier_name] = 1
# if supplier_name in products_per_supplier:
    #     current_num_products = products_per_supplier[supplier_name]
    #     current_per_suppier[supplier_name] = current_num_products + 1
    # else:
    #     print("Adding a new supplier")
    #     products_per_supplier[supplier_name] = 1

75
AAA Company
BBB Company
CCC Company
```

Ao invés de print toda a coluna ela pegou o total dado e adicionou supplier e apareceu uma linha selecionada

```
import openpyxl

inv_file = openpyxl.load_workbook("inventory.xlsx")
product_list = inv_file["Sheet1"]

products_per_supplier = {}

print(product_list.max_row)

for product_row in range(2, product_list.max_row + 1):
    supplier_name = product_list.cell(product_row, 4).value
    # print(supplier_name)

    if supplier_name in products_per_supplier:
        current_num_products = products_per_supplier[supplier_name]
```

```
        products_per_supplier[supplier_name] = current_num_products + 1
    else:
        print("Adding a new supplier")
        products_per_supplier[supplier_name] = 1

print(products_per_supplier)


75
Adding a new supplier
Adding a new supplier
Adding a new supplier
{'AAA Company': 43, 'BBB Company': 17, 'CCC Company': 14}
```



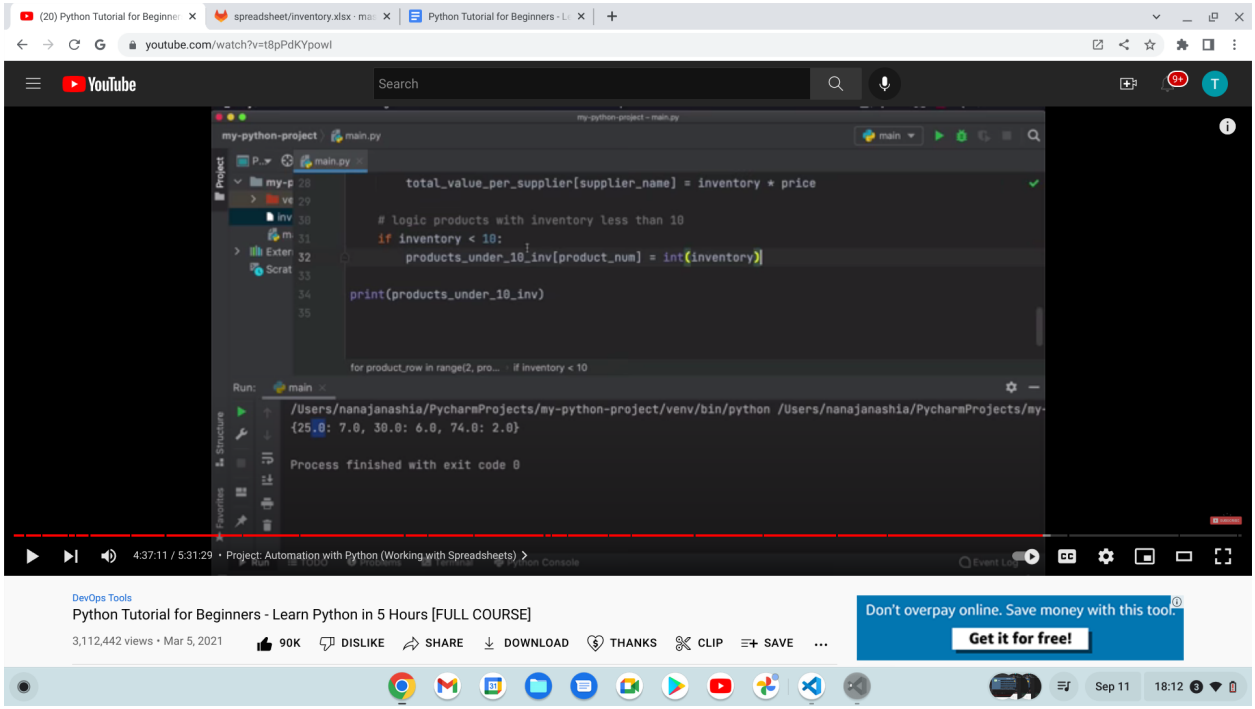

```
import openpyxl

inv_file = openpyxl.load_workbook("inventory.xlsx")
product_list = inv_file["Sheet1"]

products_per_supplier = {}

print(product_list.max_row)
```

```
for product_row in range(2, product_list.max_row + 1):
    supplier_name = product_list.cell(product_row, 4).value
    # print(supplier_name)

    if supplier_name in products_per_supplier:
        current_num_products = products_per_supplier.get(supplier_name)
        products_per_supplier[supplier_name] = current_num_products + 1
    else:
        products_per_supplier[supplier_name] = 1

print(products_per_supplier)
```

```
75
{'AAA Company': 43, 'BBB Company': 17, 'CCC Company': 14}
```

```
import openpyxl

inv_file = openpyxl.load_workbook("inventory.xlsx")
product_list = inv_file["Sheet1"]

products_per_supplier = {}
total_value_per_supplier = {}

# print(product_list.max_row)

for product_row in range(2, product_list.max_row + 1):
    supplier_name = product_list.cell(product_row, 4).value
    inventory = product_list.cell(product_row, 2).value
    price = product_list.cell(product_row, 3).value
    # print(supplier_name)

    # calculation number of products per supplier
    if supplier_name in products_per_supplier:
        current_num_products = products_per_supplier.get(supplier_name)
        products_per_supplier[supplier_name] = current_num_products + 1
    else:
        products_per_supplier[supplier_name] = 1

    # calculation total value of inventory per supplier
```

```
   if supplier_name in total_value_per_supplier:
       current_total_value = total_value_per_supplier.get(supplier_name)
       total_value_per_supplier[supplier_name] = current_total_value +
inventory * price
   else:
       total_value_per_supplier[supplier_name] =  inventory * price

print(products_per_supplier)
print(total_value_per_supplier)

{'AAA Company': 43, 'BBB Company': 17, 'CCC Company': 14}
{'AAA Company': 10969059.95, 'BBB Company': 2375499.47, 'CCC Company':
8114363.62}
```




```
import openpyxl

inv_file = openpyxl.load_workbook("inventory.xlsx")
product_list = inv_file["Sheet1"]

products_per_supplier = {}
total_value_per_supplier = {}
products_under_10_inv = {}
```

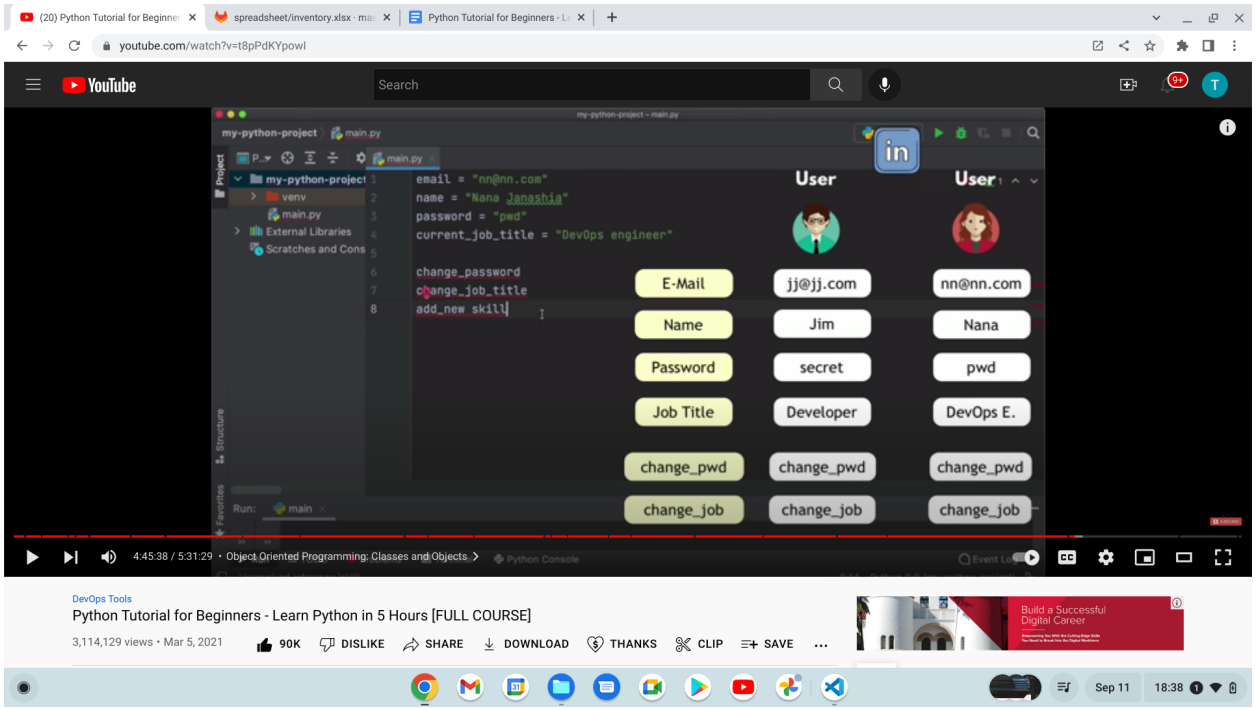
```
# print(product_list.max_row)

for product_row in range(2, product_list.max_row + 1):
   supplier_name = product_list.cell(product_row, 4).value
   inventory = product_list.cell(product_row, 2).value
   price = product_list.cell(product_row, 3).value
   product_num = product_list.cell(product_row, 1).value
   # print(supplier_name)

   # calculation number of products per supplier
   if supplier_name in products_per_supplier:
       current_num_products = products_per_supplier.get(supplier_name)
       products_per_supplier[supplier_name] = current_num_products + 1
   else:
       products_per_supplier[supplier_name] = 1

   # calculation total value of inventory per supplier
   if supplier_name in total_value_per_supplier:
       current_total_value = total_value_per_supplier.get(supplier_name)
       total_value_per_supplier[supplier_name] = current_total_value +
inventory * price
   else:
       total_value_per_supplier[supplier_name] =  inventory * price

# print(products_per_supplier)
# print(total_value_per_supplier)
   # logi products with inventory less than 10
   if inventory < 10:
       products_under_10_inv[product_num] = int(inventory)

print(products_under_10_inv)

{25.0: 7, 30.0: 6, 74.0: 2}
```

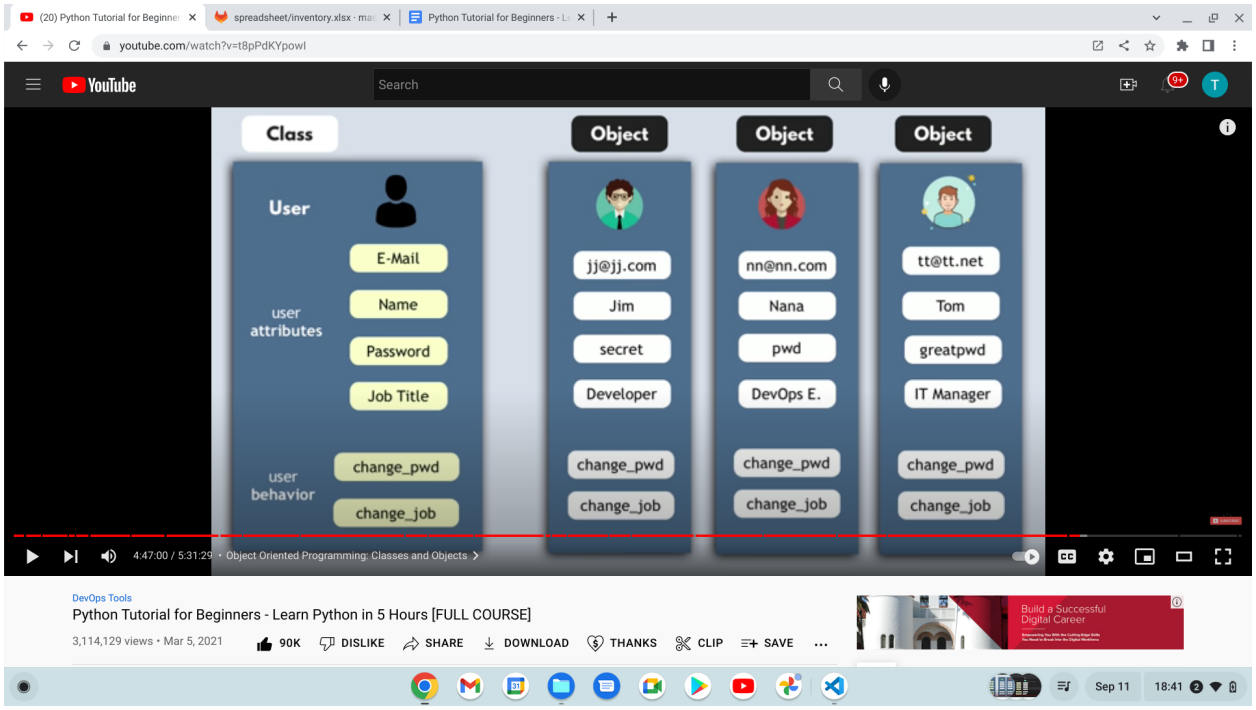
```
import openpyxl

inv_file = openpyxl.load_workbook("inventory.xlsx")
product_list = inv_file["Sheet1"]
```

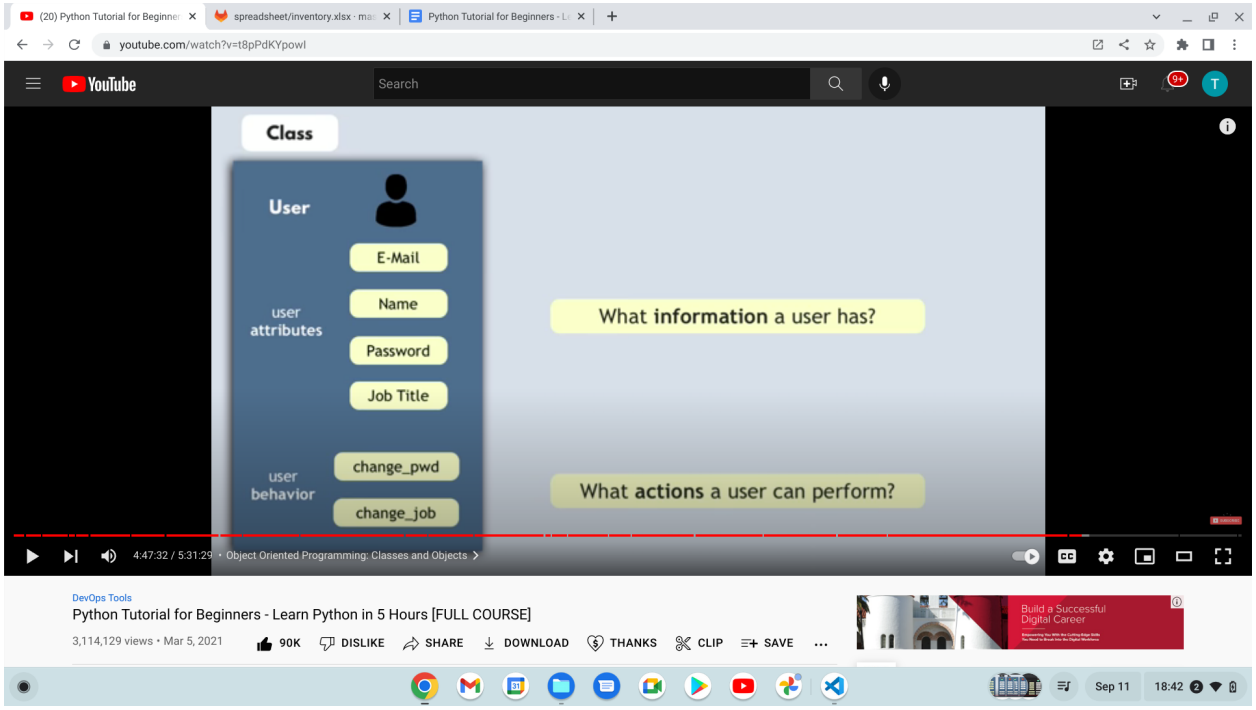
```
products_per_supplier = {}
total_value_per_supplier = {}
products_under_10_inv = {}

# print(product_list.max_row)

for product_row in range(2, product_list.max_row + 1):
   supplier_name = product_list.cell(product_row, 4).value
   inventory = product_list.cell(product_row, 2).value
   price = product_list.cell(product_row, 3).value
   product_num = product_list.cell(product_row, 1).value
   inventory_price = product_list.cell(product_row, 5)
   # print(supplier_name)

   # calculation number of products per supplier
   if supplier_name in products_per_supplier:
       current_num_products = products_per_supplier.get(supplier_name)
       products_per_supplier[supplier_name] = current_num_products + 1
   else:
       products_per_supplier[supplier_name] = 1

   # calculation total value of inventory per supplier
   if supplier_name in total_value_per_supplier:
       current_total_value = total_value_per_supplier.get(supplier_name)
       total_value_per_supplier[supplier_name] = current_total_value +
inventory * price
   else:
       total_value_per_supplier[supplier_name] =  inventory * price

# print(products_per_supplier)
# print(total_value_per_supplier)
   # logi products with inventory less than 10
   if inventory < 10:
       products_under_10_inv[product_num] = int(inventory)

   #add value for total inventory price
   inventory_price.value = inventory * price
```

```
print(products_per_supplier)
print(total_value_per_supplier)
print(products_under_10_inv)

inv_file.save("inventory_with_total_value.xlsx")

{'AAA Company': 43, 'BBB Company': 17, 'CCC Company': 14}
{'AAA Company': 10969059.95, 'BBB Company': 2375499.47, 'CCC Company':
8114363.62}
{25.0: 7, 30.0: 6, 74.0: 2}
```

Classes & Objects
Object-oriented programming - CLASSES OBJECTS



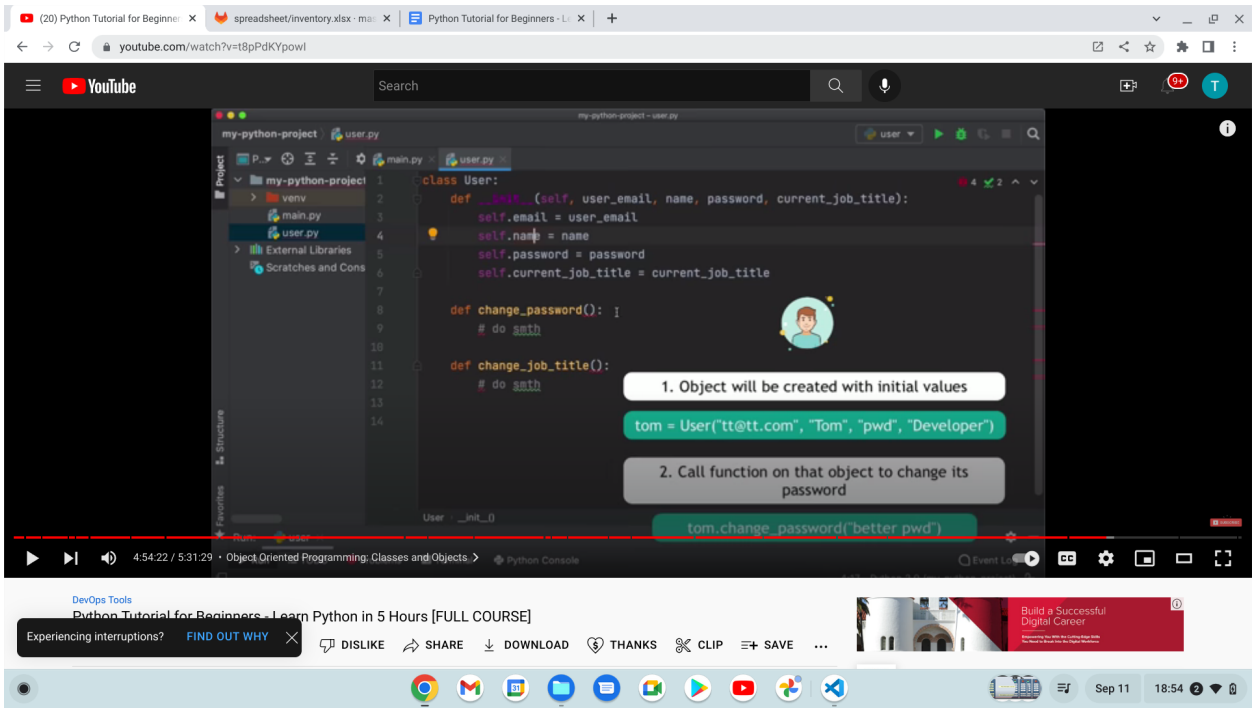





```
email = "nn@nn.com"
name = "Nana Janashia"
passowrd = "pwd"
current_job_title = "DevOps enginner"

def change_passoword():
```

```
def change_job_title():

def add_new skill():
```

Class is like an object constructor, all this have a __init__() function, __init__() is executed automatically, whenever we create the objects from this class
self is a reference to the current instance of the class, is used to acess variables that belong to the class



Class criada:

```
class User:
    def __init__(self, email, name, passoword, current_job_title):
        self.email = user_email
        self.name = name
        self.passowrd = password
        self.current_job_title = current_job_title

    def change_passoword(self, new_password):
        self.passowrd = new_password

    def change_job_title(self, new_job_title):
```

```
        self.current_job_title = new_job_title
```

Criar o objeto:
Methods: Functions that belong to an object are called methods

```
class User:
    def __init__(self, user_email, name, password, current_job_title):
        self.email = user_email
        self.name = name
        self.password = password
        self.current_job_title = current_job_title

    def change_passoword(self, new_password):
        self.password = new_password

    def change_job_title(self, new_job_title):
        self.current_job_title = new_job_title

    def get_user_info(self):
        print(f"User {self.name} currently works as a
{self.current_job_title}. You can contact then at {self.email}")

# objects
app_user_one = User("nn@nn.com", "Nana Janashia", "pwd1", "Dev0ps
Engineer")
app_user_one.get_user_info()

# app_user_one.change_job_title("Dev0ps Trainer")
# app_user_one.get_user_info()

app_user_two = User("j1@nn.com", "Jose Nashia", "pw12", "Engineer")
app_user_two.get_user_info()
```

Importando modulus

```
import user

# objects
```

```
app_user_one = user.User("nn@nn.com", "Nana Janashia", "pwd1", "Dev0ps
Engineer")
app_user_one.get_user_info()

# app_user_one.change_job_title("Dev0ps Trainer")
# app_user_one.get_user_info()

app_user_two = use.User("j1@nn.com", "Jose Nashia", "pw12", "Engineer")
app_user_two.get_user_info()
```

```
from user import User

# objects
app_user_one = User("nn@nn.com", "Nana Janashia", "pwd1", "Dev0ps
Engineer")
app_user_one.get_user_info()

# app_user_one.change_job_title("Dev0ps Trainer")
# app_user_one.get_user_info()

app_user_two = User("j1@nn.com", "Jose Nashia", "pw12", "Engineer")
app_user_two.get_user_info()
```

```
class User:
    def __init__(self, user_email, name, password, current_job_title):
        self.email = user_email
        self.name = name
        self.password = password
        self.current_job_title = current_job_title

    def change_passoword(self, new_password):
        self.password = new_password

    def change_job_title(self, new_job_title):
        self.current_job_title = new_job_title

    def get_user_info(self):
        print(f"User {self.name} currently works as a
{self.current_job_title}. You can contact then at {self.email}")
```

```
                                TERMINAL

User Nana Janashia currently works as a DevOps Engineer. You can contact
then at nn@nn.com
User Jose Nashia currently works as a Engineer. You can contact then at
j1@nn.com
```

CREATE ANOTHER CLASS AND OBJECTS

Main.py

```
from user import User
from post import Post

# objects
app_user_one = User("nn@nn.com", "Nana Janashia", "pwd1", "DevOps
Engineer")
app_user_one.get_user_info()

# app_user_one.change_job_title("DevOps Trainer")
# app_user_one.get_user_info()

app_user_two = User("j1@nn.com", "Jose Nashia", "pw12", "Engineer")
app_user_two.get_user_info()

new_post = Post("on a secret mission today", app_user_one.name)
new_post.get_post_info()
```

User.py

```
class User:
    def __init__(self, user_email, name, password, current_job_title):
        self.email = user_email
        self.name = name
        self.password = password
        self.current_job_title = current_job_title

    def change_passoword(self, new_password):
        self.password = new_password
```

```
    def change_job_title(self, new_job_title):
        self.current_job_title = new_job_title

    def get_user_info(self):
        print(f"User {self.name} currently works as a
{self.current_job_title}. You can contact then at {self.email}")
```

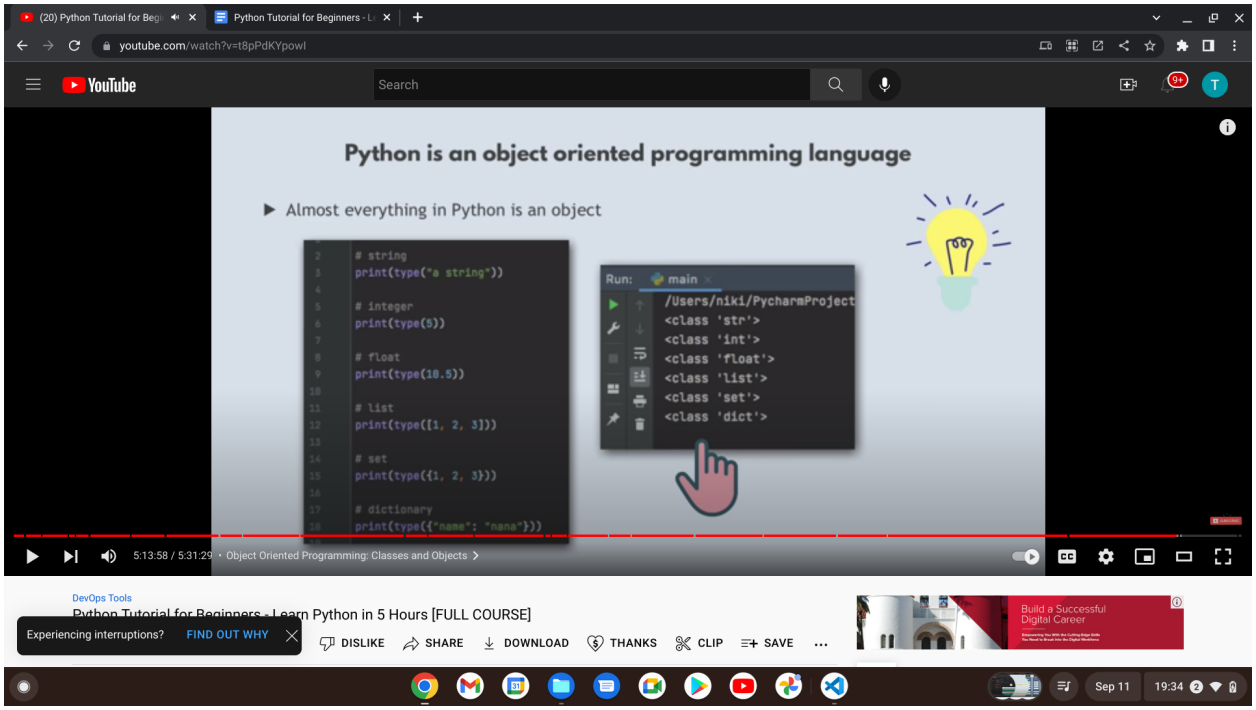
Post.py

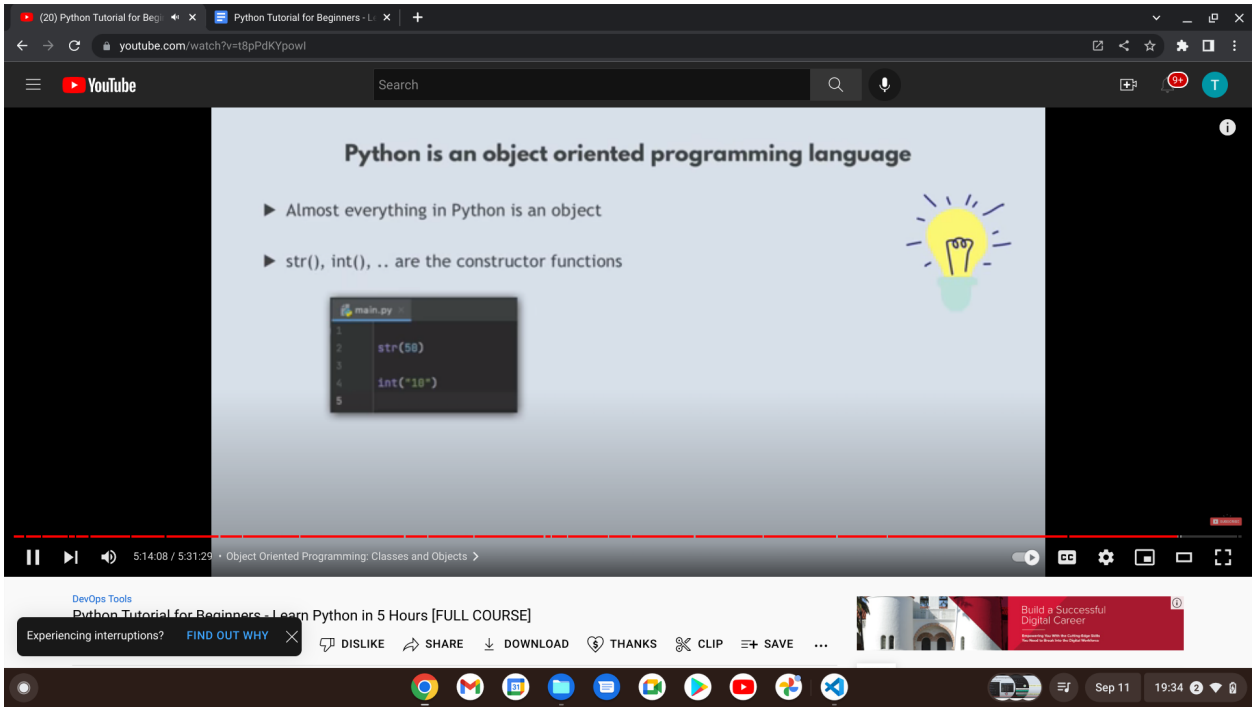
```
class Post:
    def __init__(self, message, author):
        self.message = message
        self.author = author

    def get_post_info(self):
        print(f"Post: {self.message} written by {self.author}")
```

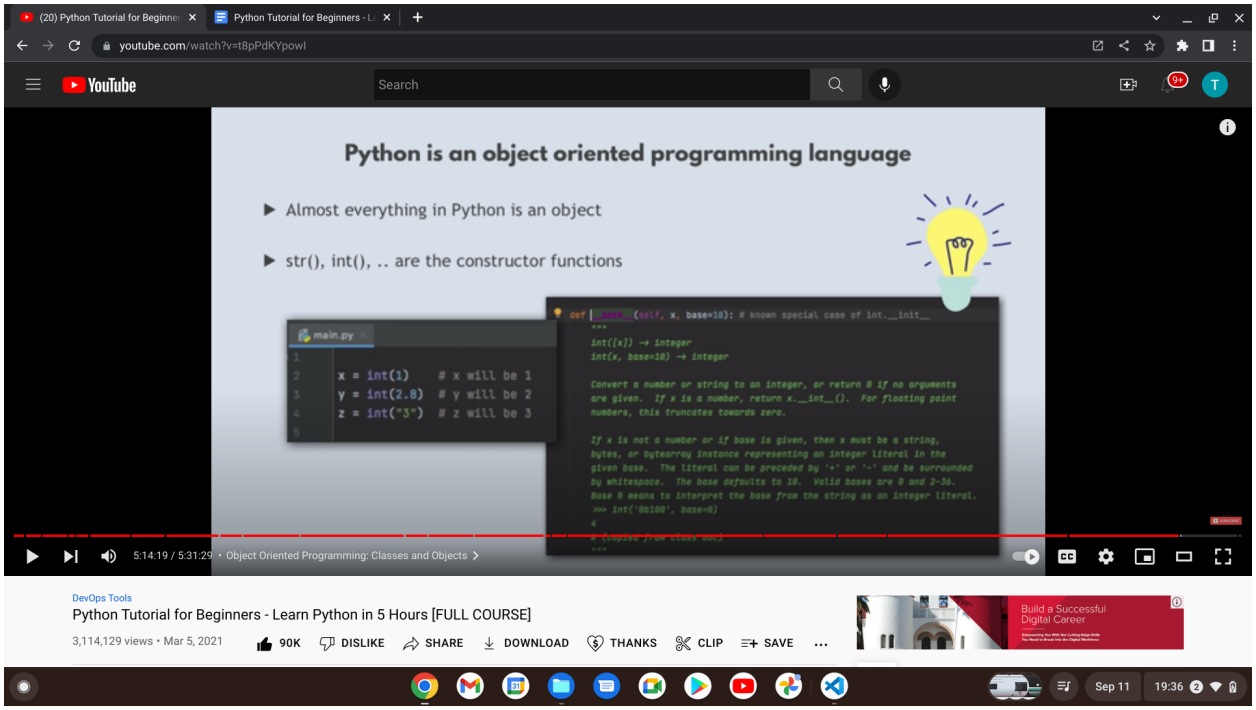
TERMINAL - python main.py

```
User Nana Janashia currently works as a Dev0ps Engineer. You can contact
then at nn@nn.com
User Jose Nashia currently works as a Engineer. You can contact then at
j1@nn.com
Post: on a secret mission today written by Nana Janashia
```

Object Oriented Programming

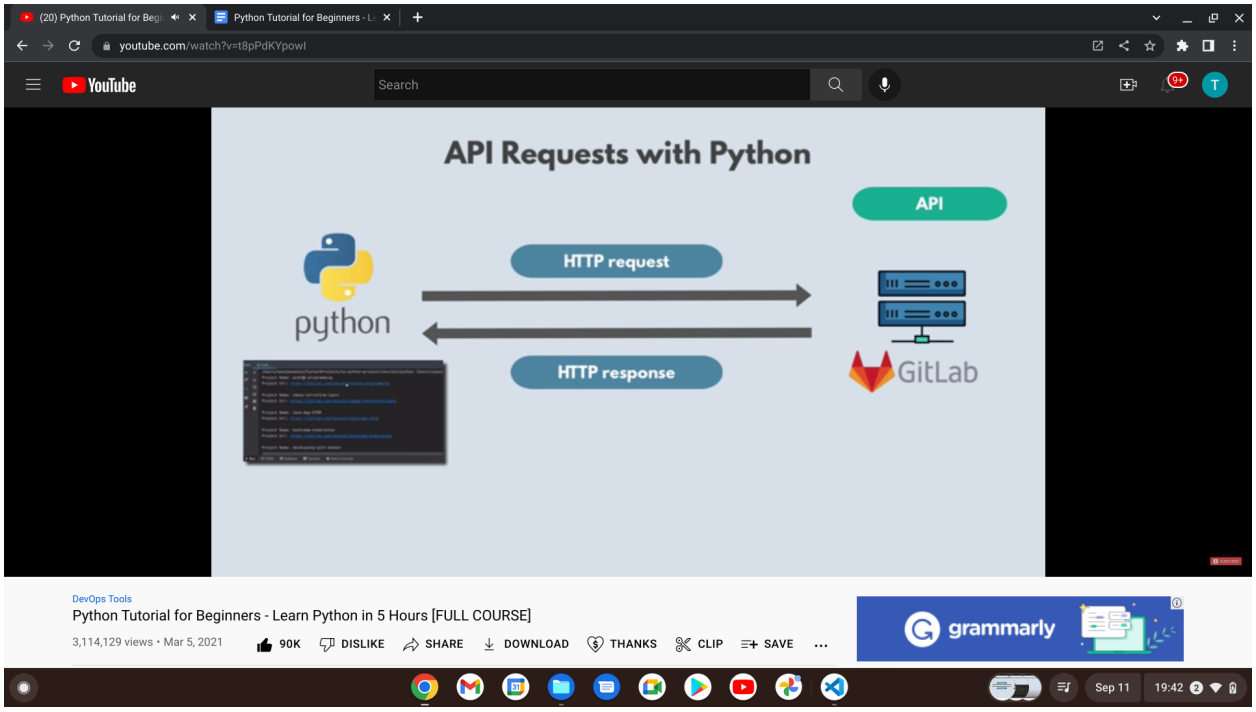

youtube.com/watch?v=t8pPdKYpowI
YouTube
Search
Python is an object oriented programming language
Almost everything in Python is an object
# string
print(type("a string"))
# integer
print(type(5))
# float
print(type(10.5))
# list
print(type([1, 2, 3]))
# set
print(type({1, 2, 3}))
# dictionary
print(type({"name": "nana"}))
Run: main
/Users/niki/PycharmProject
<class 'str'>
<class 'int'>
<class 'float'>
<class 'list'>
<class 'set'>
<class 'dict'>
5:13:58 / 5:31:29 • Object Oriented Programming: Classes and Objects
DevOps Tools
Python Tutorial for Beginners - Learn Python in 5 Hours [FULL COURSE]
Experiencing interruptions?
FIND OUT WHY
DISLIKE
SHARE
DOWNLOAD
THANKS
CLIP
SAVE
Build a Successful Digital Career
Sep 11
19:34

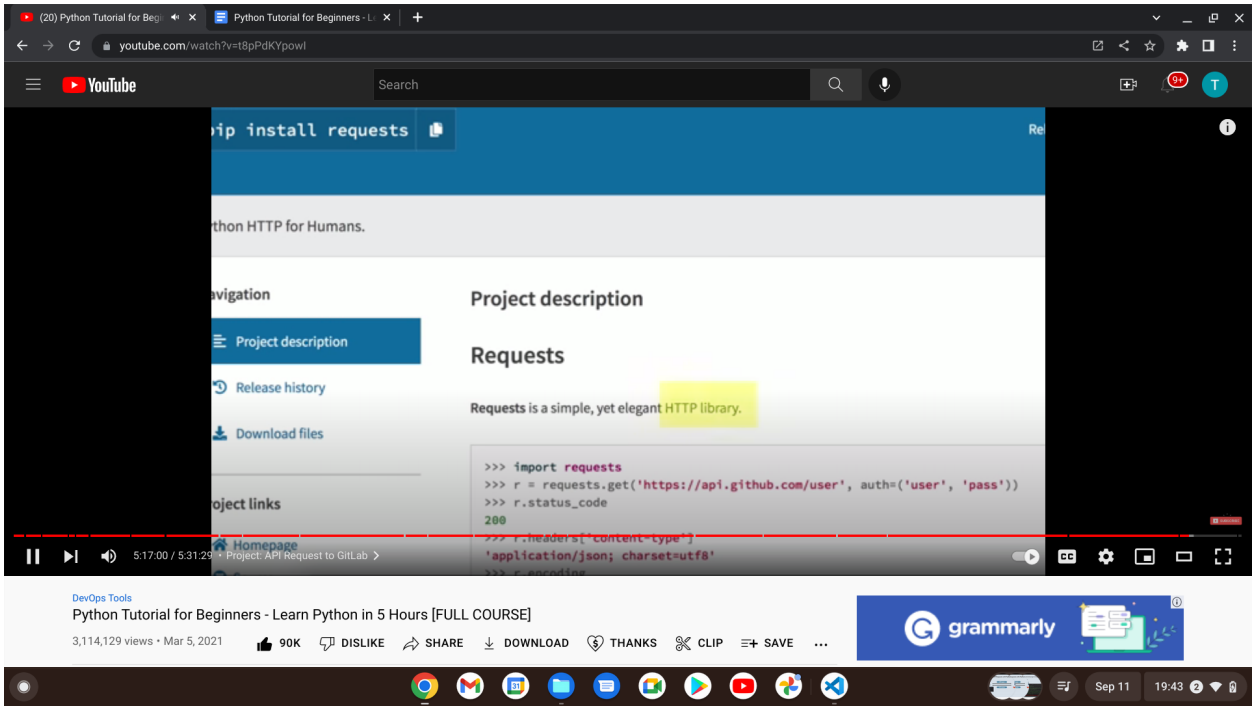

youtube.com/watch?v=t8pPdKYpowI
YouTube
Search
Python is an object oriented programming language
Almost everything in Python is an object
str(), int(), .. are the constructor functions
main.py
str(50)
int("10")
5:14:08 / 5:31:29 • Object Oriented Programming: Classes and Objects
DevOps Tools
Python Tutorial for Beginners - Learn Python in 5 Hours [FULL COURSE]
Experiencing interruptions?
FIND OUT WHY
DISLIKE
SHARE
DOWNLOAD
THANKS
CLIP
SAVE
Build a Successful Digital Career
Sep 11
19:34



Project

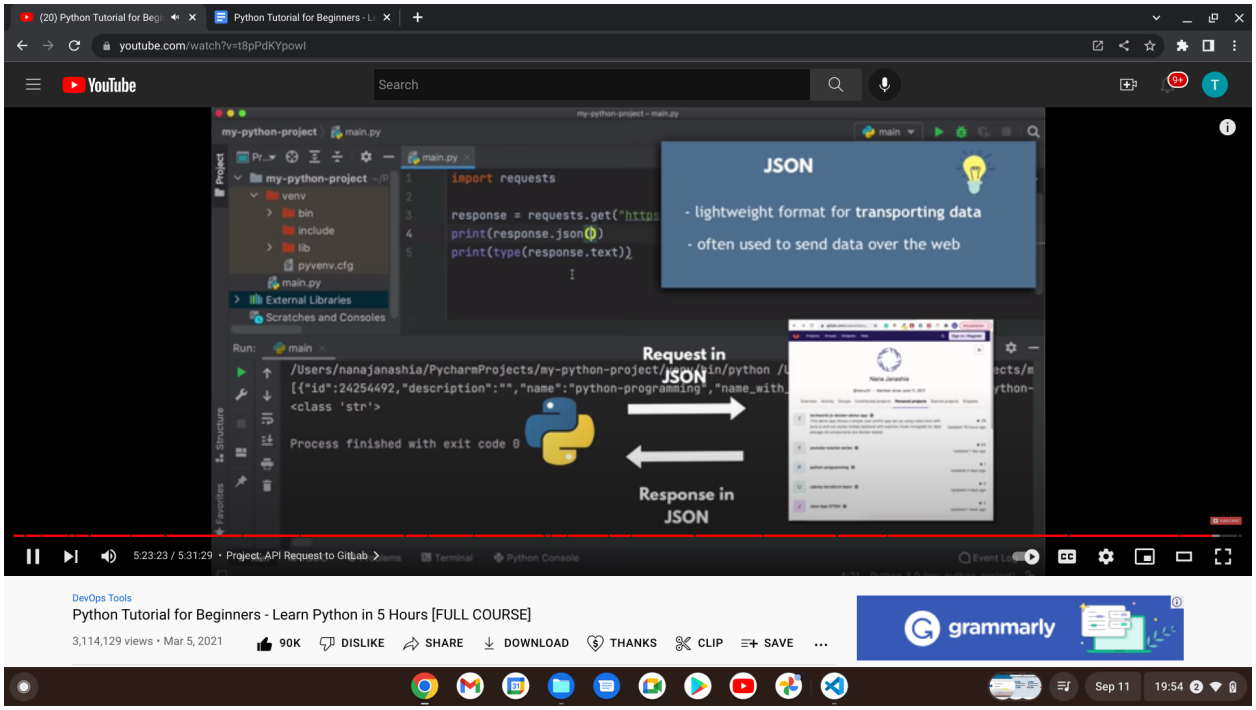
API Request to Gitlab - introduction






```python
import requests

response =
requests.get("https://gitlab.com/api/v4/users/user_id/projects")
print(response.text)
print(type(response.text))
```

Json: lightweight format for transporting data



requests json() function converts the data from json into Python data type

```
import requests

response =
requests.get("https://gitlab.com/api/v4/users/user_id/projects")
print(response.json())
print(type(response.json()))
print(response.json([0]))
```

```
import requests

response =
requests.get("https://gitlab.com/api/v4/users/user_id/projects")
my_projects = response.json()

for project in my_projects:
   print(f"Project Name: {project['name']}/n Project Url:
{projects['web_url']}/n")
```