Mathematical Constants

* math.pi
  out- 3.141592653589793
* math.e
  out- 2.718281828459045
* math.log(10)
  out- 2.302585092994046
* math.log(100,10)
  out- 2.0

Trigonometrics Functions

* math.sin(10)
  out:- 0.5440211108893698.
* math.degrees(pi/2)
  out- 90.0
* math.radians(180)
  out: 3.141592653589793
* import random
  random.randint(0,100) . always run and random value

out- 3

* random.seed(101)
  random.randint(0,100) once came then it will not change

out- 74

* mylist = list(range(0,20))

  my list

out [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

random.choice(mylist)

out- 12

Class - 21 Dt- 01/04/2023

Working with PyPDF2

for installation (pip install PyPDF2)

```
import PyPDF2
f = open('Working_Business_proposal.Pdf','rb')
Pdf_reader = PyPDF2.PdfFileReader(f)
pdf_reader.numpages
- 5
* page_one = Pdf_reader.getPage(0).extractText()
- we can then extract the text
* page_one_text = page_one.extractText()
* page_one_text  out:- (Then show the things inside Pdf)
* f.close()
```

Adding to PDFs

```
f = Open('Working_Business_Proposal.Pdf','rb')
Pdf_reader = PyPDF2.PdfFileReader(f)
first_page = Pdf_reader.getPage(0)
Pdf_writer = PyPDF2.PdFileWriter()
Pdf_writer.addPage(first_page)
Pdf_output = Open("Some_New_Doc.Pdf","wb")
Pdf_writer.write(Pdf_output)
f.close()
```

Simple Ex-

Let's try to grab all the text from this PDF file.

```
f = Open('Working_Business_Proposal.Pdf','rb')
Pdf_text = []
Pdf_reader = PyPDF2.PdfFileReader(f)
for p in range(Pdf_reader.numpages):
    page = Pdf_reader.getPage(p)
    Pdf_text.append(page.extractText())
```

* Pdf_text out - all text from pdf.

Email Sending & receiving from Python :-

```
import smtplib
* smtp_object = smtplib.SMTP('smtp.gmail.com', 587)
* smtp_object.ehlo()
out - 250 (that means connection successful)
* smtp_object.starttls()
out - (220, b'2.0.0 Ready to start TLS')
```

If 587 not work the 465

(end to end encryption) (so no one can decode)

```
import getpass
* email = getpass.getpass("Enter your email: ")
password = getpass.getpass("Enter your password: ")
smtp_object.login(email, password)
* from_address = getpass.getpass("Enter your email: ")
to_address = getpass.getpass("Enter the email of the recepient: ")
Subject = input("Enter the subject line: ")
message = input("Type out the message you want to send: ")
msg = "Subject: " + subject + '\n' + message
smtp_object.sendmail(from_address, to_address, msg)
```

out:- Enter your email: ----
Enter the email of the recepient: ----
Enter the subject line: This is a test
Type out the message you want to send: Here is the messag

Received - Emails

```
import imaplib
m = imaplib.IMAP4_SSL('imap.gmail.com')
import getpass
user = input("Enter your email: ")
password = getpass.getpass("Enter your password: ")
```

- Enter your password: ----

M.login (user, password)
M.list ( ) out: ('OK'
[b'(\\HasNoChildren) "/" "INBOX"',
" " "Personal",
"Receipts",
"Sent",
"Trash",

Connect to your inbox
M.select ("inbox")
out- ('OK', [b'28297'])

* typ, data = M.search (None, 'SUBJECT "this is a test email for python")

: typ
- 'OK'

* data
- [b'28298']

* result, email_data = M.fetch (data[0], '(RFC822)')
* raw_email = email_data[0][1]
* raw_email_string = raw_email.decode ('utf-8')

```
import email
email_message = email.message_from_string (raw_email_string)
for part in email_message.walk():
    if part.get_content_type() == "text/plain":
        body = part.get_payload (decode=True)
        print (body)
```

out: b' This is a test to see if the Python search worked.\r\n'

Class-22 Dt-02/04/2023

Creating a Decorator

```
def new_decorator (func):
    def wrap_func():
        func()
        print ("Code would be here, before executing the func", func)
        print ("Code will execute after the func();")
    return wrap_func
def func_needs_decorator():
    print ("The function is in need of a Decorator")
```

* fun_needs_ decorator ( )

out- This function is in need of a Decorator

* func_needs_ decorator = new_ decorator (func needs_ decorator)

* func_ needs_ decorator( )

out! This function is in need of a Decorator

" " " " " "

Code could be there, before executing the func None

Code there will execute after the func( ).

* @new_ decorator

def func_needs_decorator ( ):

Print ("This function is in need of a Decorator")

func_needs_ decorator( )

out:

Generator

```
def genfibon(n):
    """
    Generate a fibonnaci sequence up to n
    """
    a = 1
    b = 1
    for i in range(n):
        yield a
        a, b = b, a+b
for num in genfibon(10):
    Print(num)
```

out:

```
1
1
2
3
5
8
13
21
34
55
```

```
page_still_valid = True
authors = set()
page = 1
while page_still_valid:
    page_url = url + str(page)
    res = requests.get(page_url)
    if "No quotes found!" in res.text:
        break
    soup = bs4.BeautifulSoup(res.text, 'lxml')
    for name in soup.select(".author"):
        authors.add(name.text)
    page += 1
```

thinking.

* authors

out - {'Albert Einstein'
'Alexandre Dumans fils', ...} etc all page's names are the output

Working with Pillow Library:-

Opening images

* from PIL import Image
* mac = Image.open('example.jpg')
* type(mac)

out - PIL.JpegImagePlugin.JpegImageFile

* mac

out - Image is come

image information

* mac.size

out - (1993, 1257)

* mac.filename

out - examp.JPG

* mac.format_description

out - 'JPEG (ISO 10918)'

- authors
out- Only on authors name come {'Albert Einstein', 'Andre Gide', etc}

* quotes = [ ]
for quote in soup.select(".text"):

- quotes
out- ["The world as we have created it is a process of our thinking ..... etc]

* soup = bs4.BeautifulSoup(res.text, 'lxml')
soup.select('.tag-item')

out-

* for item in soup.select(".tag-item"):
print(item.text)

out- Love
inspirational ....etc

(Assuming You know Number of pages)

url = 'http://quotes.toscrape.com/page/'

authors = set( )

for Page in range(1, 10):
# concatenate to get new page URL
Page_url = url + str(Page)
# Obtain Request
res = requests.get(page_url)
# Turn into soup
soup = bs4.BeautifulSoup(res.text, 'lxml')
# Add Authors to out set
for name in soup.select(".author"):
author.add(name.text)

```
def simple_gen():
    for x in range(5):
        yield x

g = simple_gen()
g
```

out: <generator object simple_gen at 0x7fc351623a50>

next()

```
print(next(g))
0
print(next(g))
1
print(next(g))
2
print(next(g))
3
print(next(g))
4
```

```
* s = 'hello'
too let ints:
    print(let)
```

out- h e l l o

```
- next(s)
out- TypeError
- s_iter = iter(s)
  s_iter
out - <str_interator at 0x7fc34075c7f07
- next(s_iter)
  'h'
- next(s_iter)
  'e'
```

Web-Scraping

```
import requests
import bs4          # beautiful soup 4
res = requests.get("http://quotes.toscrape.com/")
res.text
```

out- the quotes.toscrape page open in html formate

```
soup = bs4.BeautifulSoup(res.text, 'lxml')
soup.select(".author")
```

<small class="author" itemprop="author">Albert Einstein</small>

examples

out: Ans only .author name comes from that page

```
* authors = set()
for name in soup.select(".author"):
    authors.add(name.text)
```

Cropping images

```
mac.crop(0,0,100,100))
```

Copying & Pasting images

```
computer = mac.crop((x,y,w,h))
mac.paste(im = computer, box = (0,0))
mac
```

out -

Rotate ()

```
Pencils.rotate(120)
```

out -

Class-23 Dt-03/04/2023

JSON

```
import json                                  # json - Javascript object notation
from difflib import get_close_matches        # difflib class vs data comparison
data = json.load(open("data.json"))
def translate(w):
    w = w.lower()
    if w in data:
        return data[w]
    elif w.title() in data:
        return data[w.title()]
    elif w.upper() in data:
        return data[w.upper()]
    elif len(get_close_matches(w, data.keys())) > 0:
        yn = input("Did you mean %s instead?
                    Enter y if yes, or N if no: "
                    % get_close_matches
        if yn == "y":
            return data[get_close_matches(w, data.keys())[0]
        elif yn == "N":
            return "The word doesn't exist. Please double
                    check it."
```

```
    else:
        return "We didn't understand you entry."
else:
    return "The word doesn't exist. Please double check it."
word = input("Enter word: ")
output = translate(word)
if type(output) == list:
    for item in output:
        print(item)
else:
    print(output)
```

out:- Enter word: Rome
The capital

GUI - Graphical User Interface:-

```
import ipywidgets as widgets
for item in widgets.Widget.widget_type.items():
    print(item[0][2][:-5])
```

out-

LayoutModel	DescriptionStyle	Label	vBox
Accordion	DirectionalLink	Link	valid
Audio	Dropdown	Password	video
BoundedFloatText	FileUpload	Play	Output
BoundedIntText	FloatLogSlider	ProgressStyle	
Box	FloatProgress	RadioButtons	
Button	FloatRangeSlider	Select	
ButtonStyle	FloatSlider	SelectMultiple	
CheckBox	FloatText	SelectionRangeSlider	
ColorPicker	GridBox	SelectionSlider	
comboboX	HBox	SliderStyle	
ControllerAxis	HTMLMath	Tab	
ControllerButton	HTML	Text	
Controller	Image	Textarea	
DOMWidget	IntProgress	ToggleButton	
DatePicker	IntRangeSlider	ToggleButtons	
	IntSlider	ToggleButtonsStyle	
	IntText	[illegible]	

BoundedInt Text

```
widgets.BoundedIntText(
    value = 87
    min = 0,
    max = 100,
    step = 1,
    description = 'Text:',
    disabled = False
)
```

out Text: [50 ▲▼]

BoundedFloat Text

→ same

IntText

```
widgets.IntText(
    value = 70,
    description = 'Any:',
    disabled = False
)
```

out: Any: [70 ▲▼]

float Text

same

Toggle Button

```
widgets.ToggleButton(
    value = True,
    discripton = 'Click me',
    disabled = False,
    button_styl = 'success',
#success, info, warning, danger'
    toltip = 'Description',
    icon = 'check'
)
```

out:- [✓ Click me]

Checkbox

```
widget.Checkbox(
    value = False,
    description = 'Click the button',
    disabled = False
)
```

out ☒ Click the button

valid

```
widgets.valid(
    value = False,
, description = 'Valid!',
```

out! Valid! ✗

Dropdown

```
widgets.Dropdown(
    option = ['1', '2', '3', '4', '5'],
    value = '2',
    description = 'Enter the value',
    disable = False,
)
```

out: Enter the value [5 ▼]

Drop down

```
widgets.Dropdown(
    option = {'One':1, 'Two':2, 'Three':3},
    value = 3,
    description = 'Number:',
)
```

out: Number: [Three ▼]

IntSlider

```
widgets.IntSlider(
    value = 50,
    min = 20,
    max = 80,
    step = 5,
    description = 'TEST',
    disabled = False,
    continous_update = False,
    orientation = 'horizontal',
    readout = True,
    readout_format = 'd'
)
```

out :- TEST —o— 50

FloatSlider

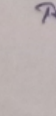
```
widgets.FloatSlider(
    value = 7.5,
    min = 0,
    max = 10.0
    setep = 0.1,
    description = 'Test:',
    disabled = False,
    continous_update = False,
    orientation = 'horizontal',
    readout = True,
    readout_format = '.1f',
```

out :- Test —o— 7.5

IntRangeSlider

```
widgets.IntRangeSlider(
    value = [3, 7],
    min = 0
    max = 10,
    step = 1,
    description = 'Test:',
    disable = False
    continous_update = False,
    orientation = 'vertical'
    readout = True,
    readout_format = 'd',
)
```

out



FloatRangeSlider

```
widgets.FloatRangeSlider(
    value = [3.7, 8.7]
    min = 0
    max = 10.0,
    step = 0.1,
    description = 'Test:',
    disabled = False,
    continous_update = False,
    orientation = 'horizontal',
    readout = True,
    readout_format = '.2f',
~~readout_format = '.2f',~~
```

out : text —o—o— 

IntProgress

```
widgets.IntProgress(
    value = 7,
    min = 0
    max = 10,
    step = 1,
    description = 'Bar',
    bar_style = 'warning',
    orientation = 'horizontal'
)
```

FloatProgress

```
widgets.FloatProgress( )
    value = 7.5
    min = 0,
    max = 10.0,
    step = 0.1,
    description = 'Loading:',
    bar_style = 'sucess',
) orientation = 'horizontal'
```

out :- Loading: 

Radio Buttons

```
widgets.RadioButtons(
    options=['Pepperoni', 'pineapple', 'anchovies'],
    value='pineapple',
    description='Pizza topping:',
    disabled=False
)
```

Pizza top...
- Pepperoni
- pineapple
- anchovies

Select

```
widgets.Select(
    option=['Linux', 'Window', 'osx'],
    value='osx',
    rows=10,
    description='OS:',
    disable=False
)
```

OS: Linux / Windows / OSX

SelectionSlider

```
widgets.SelectionSlider(
    option=['scrambled', 'sunny side up', 'poached', 'over easy'],
    value='sunny side up',
    description='I Like my eggs...';
    disabled=False,
    Continous_update=False,
    orientation='horizontal',
    readout=True
)
```

I like my egg. ——— scrambled

SelectionRangeSlide

```
import datetime
dates = [datetime.date(2015, i, 1) for i in range(1,13)]
options = [(i.strftime('%b'), i) for i in dates]
widgets.SelectionRangeSlider(
    options=options,
    index=(0, 11),
    description='Months (2015)',
    disabled=False
)
```

Months (20... 0 ——— 0 Jan-Dec

Text

```
widgets.Text(
    value = 'Hello Word',
    Placeholder = 'Type Something',
    description = 'strong:',
    disabled = False
)
```

out: strong: [ ]

Textarea

```
widget.Textarea(
    Value = 'Hello World'
    Placeholder = 'Type Something',
    description = 'Strong:',
    disabled = False
```

out: strong: [ ]