Dates, Functions, RegEx and more

Contents

- String format
- Handling Dates
- Datetime
- Functions
- args and kwargs
- Regex functions
- map() and filter()
- OOP basics

String Format for custom variables

```
avg_age=cc_usage["Customer_Age"].mean()
print("The average age is {}".format(avg_age))
```

```
avg_credit_limit=cc_usage["Credit_Limit"].mean()
print("The average age is {} and the average credit limit
is {}".format(avg_age, avg_credit_limit ))
```

One more way of formatting

```
print(f"The average age is {avg_age} , the average credit
limit is {avg_credit_limit}")
```

Reorder the Values

```
avg_Dependent_count=cc_usage["Dependent_count"].mean()
print("The average age is {} , the average credit limit is
 {} , the average Dependent count is {}".format(2.34620321
91172115, 8631.953698034848, 46.32596030413745))


print("The average age is {2} , the average credit limit i
s {1} , the average Dependent count is {0}".format(2.34620
32191172115, 8631.953698034848, 46.32596030413745))
```

Placeholders with name

```
print("The average age is {age} , the average credit limit
is {clim} , the average Dependent count is {depen}".forma
t(depen=2.3462032191172115, clim=8631.953698034848, age=46
.32596030413745))
```

Customizing the output format

```
print("The average age is {2:0.1f} , the average credit li
mit is {1:0.0f} , the average Dependent count is {0:0.1f}"
.format(2.3462032191172115, 8631.953698034848, 46.32596030
413745))
```

```
print("The average age is {age:0.1f} , the average credit
limit is {clim:0.0f} , the average Dependent count is {dep
en:0.1f}".format(depen=2.3462032191172115, clim=8631.95369
8034848, age=46.325960304137 45))
```

Print as percentage

```
avd_util_percent=cc_usage["Avg_Utilization_Ratio"].mean()
print("The Avg_Utilization_Ratio age is {:0.2f} ".format(a
vd_util_percent))
```

The Avg_Utilization_Ratio age is {:0.2f} " is a string with a placeholder for the value of "avd_util_percent".

The placeholder is indicated by "{}" and the ":0.2f" specifies that the value should be formatted as a floating point number with two decimal places.

Print as percentage

```
print("The Avg_Utilization_Ratio age is {:0.2%} ".format(a
vd_util_percent))
```

The placeholder for the value is indicated by "{}" and the ":0.2%" specifies that the value should be formatted as a percentage with two decimal places.

LAB: String Format for custom variables

- Take NY taxi fare data.
- Print the below message using string format.
  - The average taxi fare amount is XX.XX $ and the median number of passengers per trip are X
  - The percentage of trips with fare amount is greater than 50$ is X.XX%

Working with dates

Prepare the date field

```
kc_house_price=pd.read_csv("https://raw.githubusercontent.com/giridhar276/
Datasets/master/kc_house_data/kc_house_data.csv")
```

date
20141013T000000
20140611T000000
20140919T000000
20140804T000000
20150413T000000

Prepare the date field

```
kc_house_price["date"].apply(lambda x: x[0:8])
```

OR

```
kc_house_price["date_split"]=kc_house_price["date"].apply(
lambda x: x.split("T")[0])
kc_house_price["date_split"]
```

Str-p-time : strptime() from string to date

- The strptime() function is a built-in function in Python that is used to parse a string representation of a date and time and convert it to a datetime object.
- The strptime() function takes two arguments: a string representation of a date and time, and a string that specifies the format of the input string. The input string should match the specified format exactly.

```
from datetime import datetime

date_string = "2022-12-29"
date_format = "%Y-%m-%d"

date = datetime.strptime(date_string, date_format)
print(date)
```

Str-p-time : strptime() from string to date

```
from datetime import datetime
kc_house_price["date_cleaned"]=kc_house_price["date_split"
].apply(lambda x: datetime.strptime(x, "%Y%m%d"))
kc_house_price["date_cleaned"]


type(kc_house_price["date_cleaned"][0])
```

Derive Month, Day, Year, weekday

```
kc_house_price['month']=kc_house_price["date_cleaned"].apply
(lambda x: x.month)


kc_house_price['year']=kc_house_price["date_cleaned"].apply(
lambda x: x.year)


kc_house_price['day']=kc_house_price["date_cleaned"].apply(l
ambda x: x.day)


kc_house_price['weekday']=kc_house_price["date_cleaned"].app
ly(lambda x: x.weekday())
```

Extracting strings from dates

- Extract month name from date
- Extract weekday name
- Extract year in a particular format

- Is there any function that can take date time as input and extract the relevant information as strings

`strftime()`

Str-f-time : strftime() – From date to string

- The strftime() function is a built-in function in Python that is used to convert a datetime object to a string representation of a date and time.
- The strftime() function takes one argument: a string that specifies the format of the output string. The output string will be formatted according to the specified format.

```
from datetime import datetime

date = datetime(2022, 12, 29)
date_format = "%B %d, %Y"

date_string = date.strftime(date_format)
print(date_string)
```

Str-f-time : strftime() – From date to string

```
kc_house_price['month_name']=kc_house_price["date_cleaned"].apply(lamb
da x: x.strftime('%B'))
kc_house_price['month_name']

kc_house_price['weekday_name']=kc_house_price["date_cleaned"].apply(la
mbda x: x.strftime('%A'))
kc_house_price['weekday_name']

kc_house_price['year_YY']=kc_house_price["date_cleaned"].apply(lambda
x: x.strftime('%y'))
kc_house_price['year_YY']
```

Important symbols

%d: day of the month, from 1 to 31.

%m: the month as a number, from 01 to 12
%b: first three characters of the month name.
%B: full name of the month

%Y: year in four-digit format
%y: year in two-digit format

%w: weekday as a number, from 0 to 6, with Sunday being 0.
%A: weekday full string.

LAB: Dates

- Import Rossmann_sales data.
- Convert the column Date into Datetime format
- Derive these new columns
  - Month
  - Year
  - Day of the year (from 0 to 365)
  - Month full name
  - Week of the year (from 0 to 52)

Working with DateTime

```
nifty_stock_price["Datetime1"]=nifty_stock_price["Datetime"].apply(lam
bda x: x.split('+')[0])
nifty_stock_price["Datetime_cleaned"]=nifty_stock_price["Datetime1"].a
pply(lambda x: datetime.strptime(x, "%Y-%m-%d %H:%M:%S"))
nifty_stock_price["Datetime_cleaned"]
```

```
nifty_stock_price["time"]=nifty_stock_price["Datetime_cleaned"].apply(
lambda x: x.time())
nifty_stock_price["time"]
```

```
nifty_stock_price["hour"]=nifty_stock_price["Datetime_cleaned"].apply(
lambda x: x.hour)
nifty_stock_price[["Datetime_cleaned", "hour"]]
```

Working with DateTime

```
nifty_stock_price["minute"]=nifty_stock_price["Datetime_cl
eaned"].apply(lambda x: x.minute)
nifty_stock_price[["Datetime_cleaned", "minute"]]

nifty_stock_price["second"]=nifty_stock_price["Datetime_cl
eaned"].apply(lambda x: x.second)
nifty_stock_price[["Datetime_cleaned", "second"]]
```

Using strftime

```
nifty_stock_price["Datetime_cleaned"].apply(lambda x:
 x.strftime('%H'))
nifty_stock_price["Datetime_cleaned"].apply(lambda x:
 x.strftime('%M'))
nifty_stock_price["Datetime_cleaned"].apply(lambda x:
 x.strftime('%S'))
```

Duration - Time Delta

```
min_date=nifty_stock_price["Datetime_cleaned"].min()
max_date=nifty_stock_price["Datetime_cleaned"].max()
print("min_date {} max date {}".format(min_date,max_date))

#Duration in days
duration=max_date-min_date
duration

#Duration in hours
from datetime import timedelta
duration/timedelta(hours=1)

#Duration in minutes
duration/timedelta(minutes=1)
```

LAB: Datetime

- `Dataset- NewYork Taxifare sample data`
- Create the datetime variable based on pickup time
  - (hint: strip)
- Extract hour, minute and second variables.
- Which hour is the peak hour with maximum number of trips.
- Exactly how many months data is available in the dataset.
  - (hint: relativedelta)

More on Functions

Fixed number of input parameters

Finding the sum of squares of the input values

```
def sum_of_sq(a,b,c,d):
    ss_val=pow(a,2)+pow(b,2)+pow(c,2)+pow(d,2)
    return(ss_val)

sum_of_sq(a=1,b=2,c=3,d=4)
```

args

Finding the sum of squares of the input values of any length

```
def any_sum_of_sq(*args):
    ss_val=0
    for n in args:
        ss_val=ss_val+pow(n,2)
    return(ss_val)

any_sum_of_sq(1,2,3)
```

args and keywords

```
def any_sum_of_sq1(*args, power):
    ss_val=0
    for n in args:
        ss_val=ss_val+pow(n,power)
    return(ss_val)

any_sum_of_sq1(1,2,3, power=1)
any_sum_of_sq1(1,2,3, power=2)
any_sum_of_sq1(1,-2,-3, power=3)
```

args and keywords

```
def any_sum_of_sq2(*args, power, neg_numbers):
    if neg_numbers==True:
        ss_val=0
        for n in args:
            ss_val=ss_val+pow(n,power)
        return(ss_val)
    else:
        return("Negative numbers passed as input")

any_sum_of_sq2(1,-2,-3, power=3, neg_numbers=True)
any_sum_of_sq2(1,-2,-3, power=3, neg_numbers=False)
```

args and keywords

```
def any_sum_of_sq3(*args, power, neg_numbers, neg_power):
    if neg_power==True:
        print("Negative power is accepted")
    if neg_numbers==True:
        ss_val=0
        for n in args:
            ss_val=ss_val+pow(n,power)
        return(ss_val)
    else:
        return("Negative numbers passed as input")


any_sum_of_sq3(1,-2,-3, power=3, neg_numbers=True, neg_power=True)
```

*args and **kwargs

```
def any_sum_of_sq3(*args, **Kwargs):
    power, neg_numbers, neg_power=Kwargs.values()
    if neg_power==True:
        print("Negative power is accepted")
    if neg_numbers==True:
        ss_val=0
        for n in args:
            ss_val=ss_val+pow(n,power)
        return(ss_val)
    else:
        return("Negative numbers passed as input")

kwargs={"power":3, "neg_numbers":True, "neg_power":True}
args=(1,-2,-3)
any_sum_of_sq3(*args, **kwargs)
```

LAB: args and kwargs

- Create a function to that takes 4 values of the confusion matrix and calculates the accuracy
- Try to use *args instead of general parameters in the above function
- Add two keyword parameters recall= True, precision=True and include those two metrics in the output
- Use *args and **kwargs while defining the function

RegEx functions

len()

```
len(air_bnb_ny["name"])
```

startswith() and endswith()

```
mask=air_bnb_ny["name"].str.startswith("Affordable")
air_bnb_ny[mask]
```

```
mask=air_bnb_ny["name"].str.startswith("Affordable", na=False)
air_bnb_ny[mask]
```

```
mask=air_bnb_ny["name"].str.endswith("hotel", na=False)
air_bnb_ny[mask]
```

find()

```
"Entire home/apt".find("apt")

"small room".find("apt")

air_bnb_ny["room_type"].value_counts()
mask=air_bnb_ny["room_type"].str.find("Shared")!= -1
air_bnb_ny[mask]
```

split()

```
"Entire home/apt".split()
"Entire home/apt\tother".split()
"Entire home/apt\tother\nNA".split()
"Entire home/apt\tother\nNA".split("/")

air_bnb_ny["host_name"].str.split("&") # Two Hosts

air_bnb_ny["host_name"].str.split("&")[0] # First Host
air_bnb_ny["First_host"]=air_bnb_ny["host_name"].str.split("&").str.get
(0)
air_bnb_ny["First_host"]

air_bnb_ny["host_name"].str.split("&").str.get(1) # Second Host
```

Contains()

```
mask=air_bnb_ny["name"].str.contains("only for", na=False)
air_bnb_ny[mask]
```

replace()

```
air_bnb_ny["room_type_new"]=air_bnb_ny["room_type"].str.re
place("Entire home/apt", "Guest House")
air_bnb_ny["room_type_new"].sample(20)
```

```
air_bnb_ny["room_type_new"]=air_bnb_ny["room_type"].str.re
place("entire home/apt", "Guest House", case=False)
air_bnb_ny["room_type_new"].sample(20)
```

More on Regex

- Identify email address inside text
- Identify the numbers inside text
- Identify html tags inside text
- Identify specific repeating patterns inside text

Commonly used regular expressions

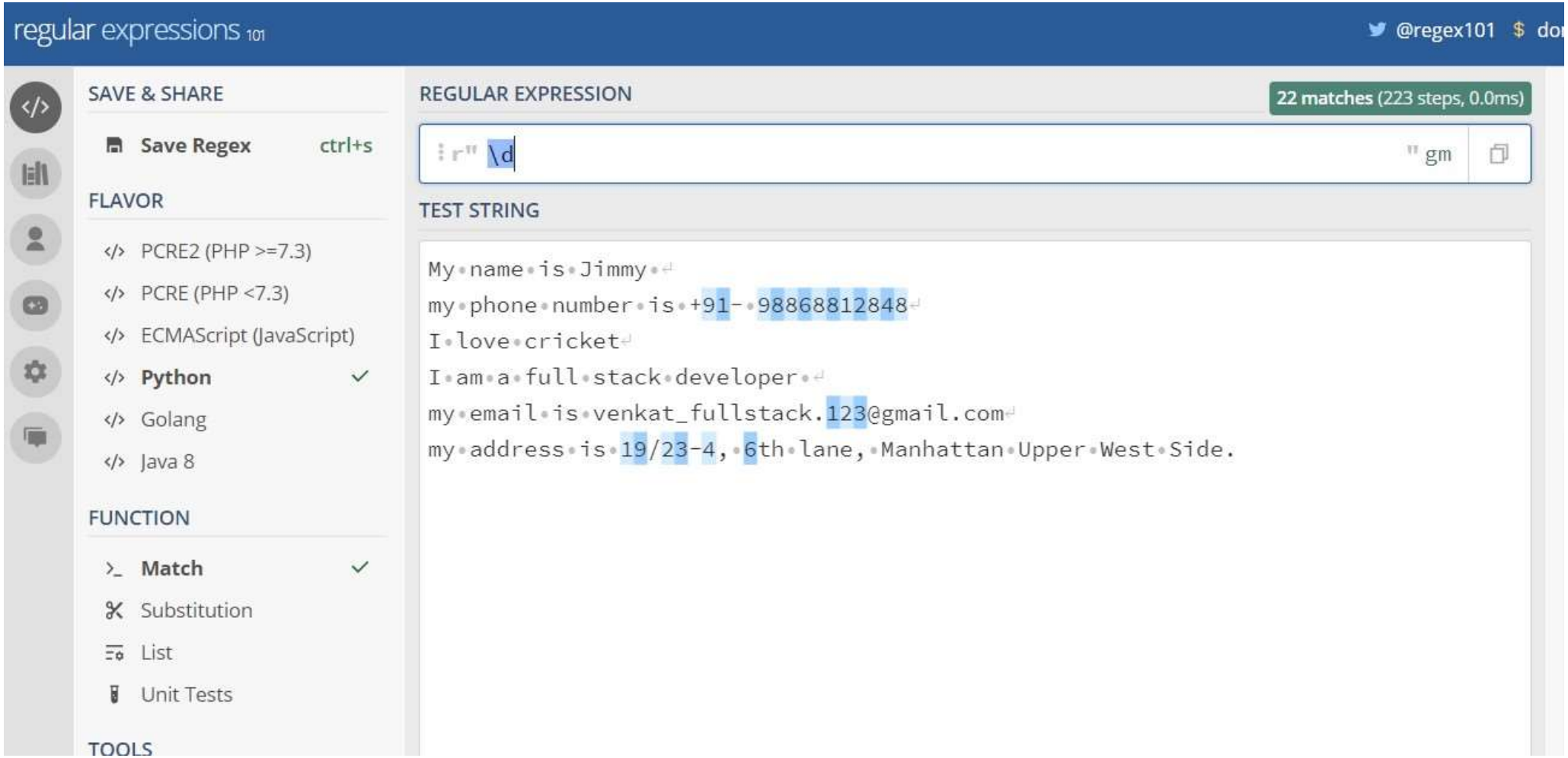
- Digits
  - Whole Numbers – /^\d+$/
  - Decimal Numbers – /^\d*\.\d+$/
  - Whole + Decimal Numbers – /^\d*(\.\d+)?$/
  - Negative, Positive Whole + Decimal Numbers – /^-?\d*(\.\d+)?$/
  - Whole + Decimal + Fractions – /[-]?[0-9]+[,.]?[0-9]*([\/][0-9]+[,.]?[0-9]*)*/
- Alphanumeric Characters
  - Alphanumeric without space – /^[a-zA-Z0-9]*$/
  - Alphanumeric with space – /^[a-zA-Z0-9]*$/
- Email
  - Common email Ids – /^([a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6})*$/
  - Uncommon email ids – /^([a-z0-9_\.\+-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/
- URLs
  - /https?:\/\/(www\.)?[-a-zA-Z0-9@:%._\+~#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+.~#()?&//=]*)/

RegEx Basic Symbols

REGEX BASICS	DESCRIPTION
^	The start of a string
$	The end of a string
.	Wildcard which matches any character, except newline (\n).
|	Matches a specific character or group of characters on either side (e.g. a|b corresponds to a or b)
\	Used to escape a special character
a	The character "a"
ab	The string "ab"

regex101.com

https://regex101.com/



Regex

```
import re
twitter_data["raw_removed_digits"]=twitter_data["raw_tweet
"].apply(lambda x: re.sub(r"\d+"," ",str(x))) #Remove digi
ts
twitter_data.iloc[300:350]
```

map() function

map() function

- The map() function is a built-in function in Python that applies a function to each element of an iterable (such as a list, tuple, or string) and returns a new iterable with the modified elements.
- The map() function takes two or more arguments: a function and one or more iterables. It applies the function to each element of the iterables, and returns a new iterable with the modified elements.

```
def add_five(x):
    return(x + 5)

numbers = [1, 2, 3, 4, 5]
result = map(add_five, numbers)
print(list(result))
```

map() function

```
mask=air_bnb_ny["name"].str.contains("only for", na=False)
sp_names=set(air_bnb_ny["name"][mask])
sp_names

def clean_tile(x):
    return(x.replace("only for", " "))

result=map(clean_tile, set(sp_names))
print(list(result))
```

map() function

```
result=map(lambda x: clean_tile(x), sp_names)
print(list(result))
```

Filter function

filter()

- The filter() function is a built-in function in Python that filters an iterable (such as a list, tuple, or string) by applying a function to each element and returning only the elements that return a True value.

- The filter() function takes two arguments: a function and an iterable. It applies the function to each element of the iterable, and returns a new iterable with only the elements that return a True value when the function is applied to them.

```
def is_even(x):
    return x % 2 == 0

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
result = filter(is_even, numbers)
print(list(result))
```

filter()

```
def is_long_word(word):
    return len(word) > 5

words = ['apple', 'banana', 'cherry', 'date', 'elderberry',
'fig']
long_words = filter(is_long_word, words)
print(list(long_words))
```

Note- You can use the filter() function with any function that takes an element as an input and returns a Boolean value (True or False). The function is applied to each element of the iterable, and only the elements that return a True value are included in the resulting iterable.

filter()

```
all_names=air_bnb_ny["name"]

def clean_tile_f(x):
    result=str(x).find("only for")!= -1
    return(result)

result1=filter(clean_tile_f, set(all_names))
print(list(result1))
```

filter()

```
result1=filter(lambda x: clean_tile_f(x), all_names)
print(list(result1))
```

Object Oriented Programming

Using the sk-learn package

```
import sklearn
model_1 = sklearn.linear_model.LinearRegression()
model_1.fit(X_train, y_train)
```

This is a model object

The internal code of sk-learn

Linear Regression Class

```
class LinearRegression(MultiOutputMixin, RegressorMixin, LinearModel):

    def __init__(self, *, fit_intercept=True, normalize=False, copy_X=True,
                 n_jobs=None, positive=False):
        self.fit_intercept = fit_intercept
        self.normalize = normalize
        self.copy_X = copy_X
        self.n_jobs = n_jobs
        self.positive = positive


        def fit(self, X, y, sample_weight=None):

        X, y = self._validate_data(X, y, accept_sparse=accept_sparse,
                                   y_numeric=True, multi_output=True)

        if sample_weight is not None:
            sample_weight = _check_sample_weight(sample_weight, X,
                                                 dtype=X.dtype)

        X, y, X_offset, y_offset, X_scale = self._preprocess_data(
            X, y, fit_intercept=self.fit_intercept, normalize=self.normalize,
            copy=self.copy_X, sample_weight=sample_weight,
            return_mean=True)

        if sample_weight is not None:
            # Sample weight can be implemented via a simple rescaling.
            X, y =  rescale_data(X, y, sample_weight)
```

Procedural programming

- If we have to repeat the same task multiple times, we create a function with several parameters.
- We can call the function and pass the relevant parameters to output our results.
- The above method is known as “procedural programming”

Procedural programming

- For example, if we are writing a function performing regression related tasks. We create a large regression function and some sub-functions in it.
  - `def regression(x, y)`
    - `Function-1 : beta coeffects`
    - `Function-2 : R-squared`
    - `Function-3 : VIF`

Object-Oriented Programming (OOP)

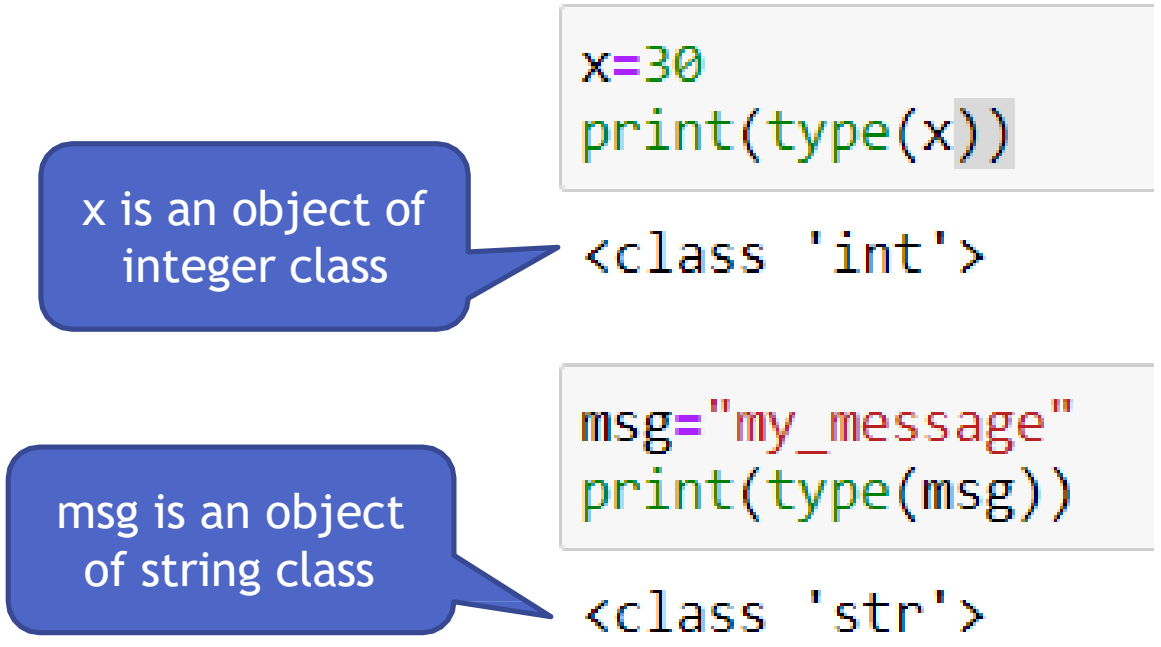
- OOP is a different programming paradigm
- Instead of writing a function, we can create classes and we can structure the large programs in a better way

Why OOP is used in development ?

- Easier to organize large programs
- Improves programs readability
- Improves software reusability
- Makes programs easier to develop.

Class and object

- Class
  - A class is like a blueprint or a design
  - In a class we have methods(functions) to perform the calculations
  - int and str are classes
- Object
  - An instance of the class is called an object
  - Object is created using the class blueprint
  - x and msg are objects



Class and objects

- Class
  - int and str are classes

- Object
  - x and msg are objects

```
x=30
print(type(x))
```

```
<class 'int'>
```

x is an object of integer class

```
msg="my_message"
print(type(msg))
```

```
<class 'str'>
```

msg is an object of string class

Class and objects

Class	Object	Description
Person	alice	Represents a person with a name and age
Dog	buddy	Represents a dog with a name, breed, and age
Car	toyota	Represents a car with a make, model, and year
Student	jane	Represents a student with a name, ID, and course
Employee	bob	Represents an employee with a name, ID, and job title

Class and objects

- In the above examples, the classes (Person, Dog, Car, Student, Employee) define the properties and behaviour of the objects (alice, buddy, toyota, jane, bob), which are instances of those classes.
- For example, the "Person" class might have attributes like "name" and "age", and methods like "speak()" and "introduce()". The "alice" object would be an instance of the "Person" class, with the name attribute set to "Alice" and the age attribute set to 25.
- Similarly, the "Dog" class might have attributes like "name", "breed", and "age", and methods like "bark()" and "play()". The "buddy" object would be an instance of the "Dog" class, with the name attribute set to "Buddy", the breed attribute set to "Labrador", and the age attribute set to 2.

Object oriented programming

- For example, if we are writing a function performing regression related tasks. We create a regression class and define methods in it.
  - `Class regression:`
    - `Properties`
      - `Input data name`
      - `Data Size`
      - `Other details`
    - `Methods`
      - `Method-1 : beta coefficients`
      - `Method-2 : R-Squared`
      - `Method-3 : VIF`

Defining a class and object

- `self` - Rerefers to the object itself. We need to pass the object itself as the input every time we are using the object
- `__init__`is a special method. It initializes the attributes of the class.

```
class any_point:
    def __init__(self, x_cord, y_cord):
        self.x=x_cord
        self.y=y_cord
        print("this point = (", self.x, self.y, ")")
```

Define objects

- `point1=any_point(0,0)`
- `point2=any_point(4,5)`

methods inside a class

```
class any_point:
    def __init__(self, x_cord, y_cord):
        self.x=x_cord
        self.y=y_cord
        print("this point = (", self.x, self.y, ")")

    def eu_dist(self, new_point):
        dist=((self.x-new_point.x)**2+(self.x-new_point.x)**2)**0.5
        return dist

    def man_dist(self, new_point):
        dist1=abs(self.x-new_point.x)+abs(self.x-new_point.x)
        return dist1
```

Using the methods from a class

```
point1=any_point(0,0)
point2=any_point(4,5)

point1.eu_dist(point2)
point1.man_dist(point2)
```

Inheritance

- Inheritance is the ability to derive or inherit the properties from another class.

Inheritance

```
class midpoint:
    def __init__(self, x_cord, y_cord):
        self.x=x_cord
        self.y=y_cord
        print("this point = (", self.x, self.y, ")")

    def mid_point(self, new_point):
        mid_x=(self.x+new_point.x)/2
        mid_y=(self.y+new_point.y)/2
        return (mid_x, mid_y)
```

Inheritance

```
class midpoint1(any_point):
    def mid_point(self, new_point):
        mid_x=(self.x+new_point.x)/2
        mid_y=(self.y+new_point.y)/2
        return (mid_x, mid_y)
m=midpoint1(0,0)
n=midpoint1(4,5)
m.mid_point(n)
```

Thank you