Introduction to Programming for Data Analysis

Using python.

Iván Andrés Trujilllo Abella

Facultad de Ingenieria
Pontificia Universidad Javeriana

trujilloiv@javeriana.edu.co

Basic structures

- list
- tuples
- dictionary

The list dont keep the object by itself by otherwise keep the memory direction to itself therefore we need take account when copy a list.

Basic structures

Implementation

```
lista = []
tupla = ()
dictionary={}
```

Basic structures

Methods are actions over in a list we can inssert o move objects

strings

Strings,

Python Structure

LIST

list are a simple structure to save elements of several data types: numeric, boolans, integers and so, even save another list.

bisection search

This method reduce in the worst case to a $O(log(n))$ complexity.

Proof.

if we have a array of k, then in the first iteration we have $k/2$ elements, if not end, then $k/4$, and so on until $k/2^p$ iterations until p reach the number, therefore. □

Base code

Bisection search

We need search a number in a sorted list.

```
if longitude of l==0:
    return False
elif longitude of l==1:
    return True
else:
    mid = longitude l / 2
print("Mathematical Background")
# This is a good method to implement
```

list

Empty string

list it is a built -in.

```
list = []
```

and empty string, we can focus in several elements.

```
list=[1,2,True,'circle']
```

Adding one element,

```
list.append('String')
```

Iterable object

list

An iterable object means that we can 'run' over each element that compound in python this start off in 0 index.

```
lista = ['A','B','C']
for j in lista:
    print(j)
```

a list of lists.

```
ListaS=[[1,2,3],[4,5,6],[7,8,9]]
i = 0
for j in listaS:
  i = i+1
  print(j,'row',i,'\n')
```

Iterable object

Dictionary

we we iterate over a dictionary we iterate over the **key value**

```
info = {'key_one':'1', 'key_two':'2'}
```

```
for v in info:
    print(v)
```

this will return:

```
key_one
key_two
```

Iterable object

Dictionary

to access to dictionary values:

```
dco = {'Key_one':'Triangle, square, circle','Key_two':'red,
    green, black'}
dco['Key_one']
```

Will return the string

```
'triangle,square,circle'
```

take in mind the that dictionary also could keep another types of data that are very useful joint the methods.

Dictionary

```
dco = {'key_one':['Triangle, square, circle'],'2':'red, green,
    black'}
dco['key_one'].append('rectangle')
```

Take in mind that python it is based upon the object oriented programming we need take in mind the data type that return python in this case the list have the attribute append and this operation is allowed. The **type(object)** could be very useful to check de data type.


Pontificia Universidad
JAVERIANA
Bogotá

indexing

Sometines it is need entry to

Example 1

Adding $i-th$ elements in list

```
matrix = [[0,2,10,30],[3,5,4,10],[0,1,2,3],[10,11,12,14]]
pairs=[]
sum=0
for x in range(len(matrix[0])):
    print(sum)
    pairs.append(sum)
    sum=0
    for j in range(len(matrix)):
      #print(j,x)
      sum = matrix[j][x]+sum
pairs.append(sum)
pairs = pairs[1:]
print(pairs)
```

```
mat = [[1,2,3,12,33],[4,5,6,0,1],[0,1,0,0,0]]
col = len(mat[0])
sol = []
for k in range(col):
  counter = 0
  for j in mat:
    counter = counter + j[k]
  sol.append(counter)
print(sol)
```

Example 2

ordering each list inverse fix the algorithm

```
new=[]
k= 1
for x in lista:
  row=[]
  j= (len(x) -1)
  while (j)!=-1:
    row.append(x[j])
    j = j -1
  k=k+1
  new.append(row)
print(new)
```

```
lista = [13,44,12,5]
new=[]
row = []
last = len(lista) - 1
while last!=-1:
  new.append(lista[last])
  last = last -1
```

Functions

The **Functions** are very handy to work with repetitive process.

```
def Function_name(k,f,h ...):
    statements
    return result
```

The example of function it is

```
def root(n,k):
    return n**(1/k)
```

Factorial function

There are different ways of compute the same thing, the factorial it is a practical function in combinatorial analysis and statistics.

$$\begin{aligned} n! &= n(n-1)(n-2)(n-3)...1 \\ n! &= n(n-1)! \end{aligned} \tag{1}$$

For loop

Factorial function

the built in function type().

```
def factl(n):
    prod = 1
       for x in range(1,n):
       prod *= x
    return prod
```

While loop

Factorial function

```
def fac2(n):
    prod = 1
    while n > 1:
        prod *= n
        n -= 1
    return prod
```

Recursion

It is a method to tackle problems that contain instance of a smaller problem of itself.

```
def fact(n):
    if n==1:
        return 1
    else:
        return n*fact(n-1)
```

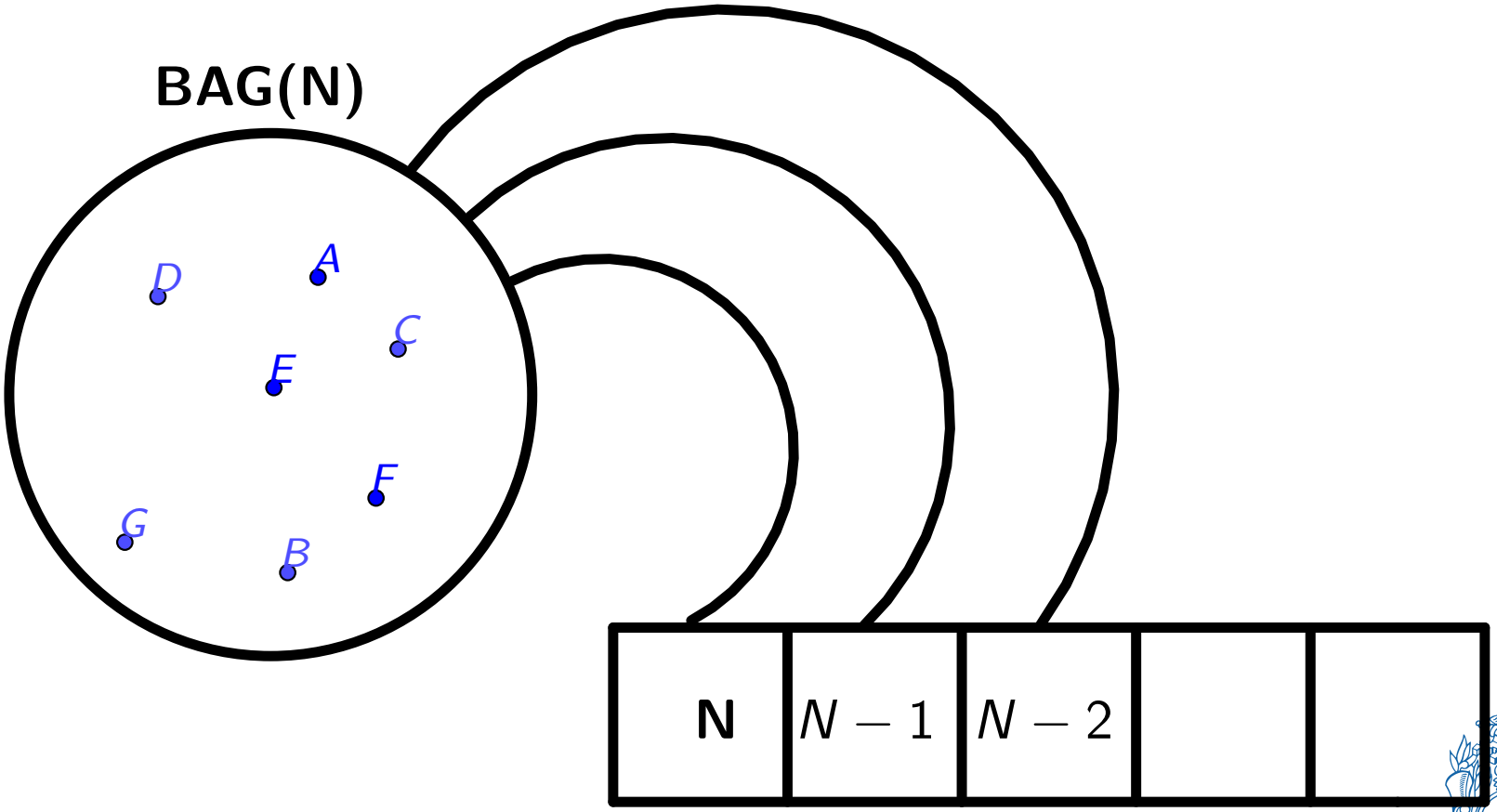
Combinatorial insights

The way in which we can order a arrangement it is important to solve some problems. **Multipliation rule**

Bag model

Suppose that you have a Bag and inside of this bag there are N balls, therefore the experiment will consist in draw some balls and write the outcome.

Bag model ilustration



Permutation - Combination

in a permutation there is not replacement and order matter.

$$P_k^n = N(N-1)(N-2)(N-3)...(N-k+1) \tag{2}$$

However for each arragement of k longitude there are $k!$ of each arragement, therefore the number of possible ways of different total elements it is a combination.

$$C_k^n = \frac{P_k^n}{k!} = \frac{n!}{(n-k!)k!} \tag{3}$$

Variation

According to the bag model suppose that there is replacement or we come back the ball to the bag,

$$V_k^n = n^k \tag{4}$$

Binomial Theorem

Binomial theorem have a refined symmetry, related with pascal's triangle.

$$(a+b)^2 = a^2 + 2ab + b^2$$

you will see that if you put n-power then will be n+1 terms. this means if $(a+b)^k$ then we will get $k+1$ terms.

$$(a+b)^n = a^n + na^{n-1}b + \frac{n(n-1)a^{n-2}b^2}{2!} + \frac{n(n-1)(n-2)a^{n-3}b^3}{3!} + \ldots + \frac{n(n-1)(n-2)(n-3)\ldots(n-r+2)a^{n-r+1}b^{r-1}}{(r-1)!} + \ldots + nab^{n-1} + b^n$$

the r term;

$$\frac{n(n-1)(n-2)(n-3)\ldots(n-r+2)a^{n-r+1}b^{r-1}}{(r-1)!} \quad (6)$$

Pascal triangle

The binomial coefficients are combinatory number. A useful fact is that in the next figure know like the pascal triangle this are in order.

$n = 0$: 1

$n = 1$: 1 1

$n = 2$: 1 2 1

$n = 3$: 1 3 3 1

$n = 4$: 1 4 6 4 1

$n = 5$: 1 5 10 10 5 1

Probability

Naive definition

given a experiment we can produce a Ω set that contain all possible outcomes **related** to the experiment.
The naive definition it is *recursive* due imposed the condition that all events are equal likelihood.

Kolgomorov approach

The certainty, it is one (1), for instance; the following statement *A person is live or dead it is certainty*

$$P(A \cup A^c) = 1 \tag{7}$$

Could be a natural property that two two two events mutually exclusive could be sum up their probabilities.

$$P(A \cup B) = P(A) + P(B) \tag{8}$$

You can extend this properties to n events.

Birthday paradox

Programming and mathematics are practical. Statistics it is a field in which we can interact with real data, the **birthday paradox** it is a practical example of this.
If we have k persons then the probability that two of them born in the same day is; if the year have 360 days:

$$1 - \frac{360.359.358...(360 - k + 1)}{360^k}$$

In the case of :

$$1 - \left(\prod_{i=311}^{360} i(360^{-50}) \right) \approx 0,97 \tag{9}$$

is 97% is a high probability that at least two person born the same day.

The following implementation is based on the uses of factorials, that are defined in *fact*() in a recursive way.

```
def permutation(n,k):
  return fact(n) / fact(n-k)

def paradox(n,k):
  return 1-(permutation(n,k)/(n**k))
```

The following Figure 92 show the relation among the probability of two coincidences among two person in the sample.

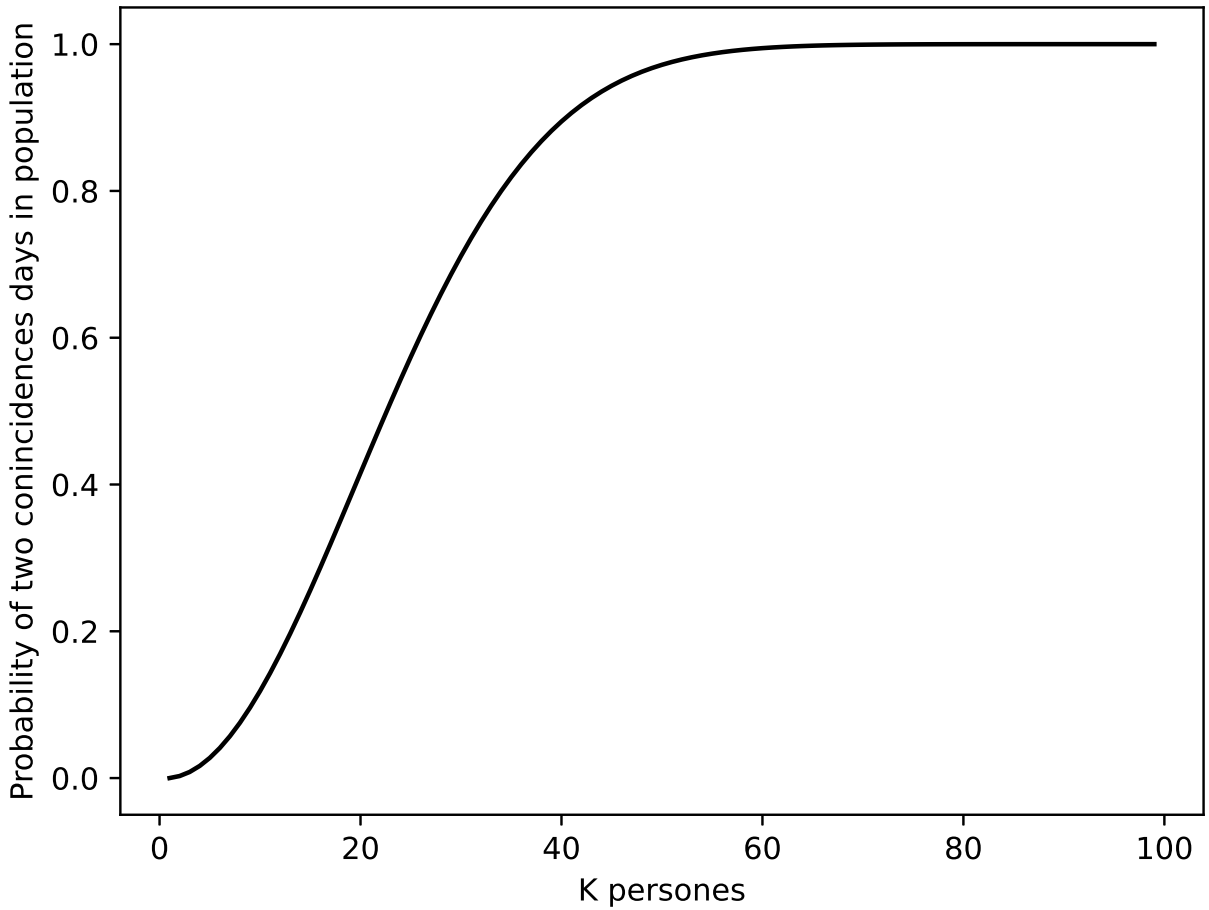


Figure: Birthday paradox with 360 days

# of 1-1, 2-2,... k-k

The problem consist in estimate the total number of combinations of genes, therefore:

$$\sum_{i=1}^{n} \binom{n}{i} \tag{10}$$

however implementing this require n computations of *fact* or *factl* defined previously, notice that $\sum_{i=0}^{n} \binom{n}{i} = 1 + \sum_{k=1}^{n} \binom{n}{k}$.

$$(x+y)^n = 1 + \sum_{k=1}^{n} \binom{n}{k} x^{n-k} y^k \tag{11}$$

if we suppose that x and y, both are equal to one, then.

$$2^n - 1 = \sum_{i=1}^{n} \binom{n}{k} \tag{12}$$

Geometric Distribution

Model the number of failures that we need to carry out until reach a success. For instance, the number of trials with vaccination needed to reach that the vacumn be useful.

$$P(X = x) = (1 - p)^{x-1}p \tag{13}$$

How many test we must carry out until the first infected patient will be selected.

```
def geometric(p,x):
  if p>1 or p<0:
    print('P must be a probability')
  else:
    return ((1-p)**x)*p
```

list comprehension

Composed of expression, if - clauses, and loops.

```
[exp for i in iterable]
odds = [["par",i] if i%2==0 else ["impar",i] for i in
    range(1,11)]
print(odds)
```

Geometric Comparation

```
def geometric(p,x):
  if p>1 or p<0:
    print('P must be a probability')
  else:one
    return ((1-p)**x)*p
p = [ 3, 12, 24, 33]
k=22
for x in p:
  exec(f"p_{x} = [geometric({x/100},i) for i in range(k)]")
  exec(f"plt.plot(np.arange(k),p_{x})")
```

Geometric CDF

Cumulative Distribution Function

The question it is $P(X > x)$.

$$CDF = 1 - (1 - p)^k \tag{14}$$

Memoryless

Geometric Distribution

Past events not affect future. In the discrete time the geometric have this property and in continous the exponential distribution too. They are related?

Theorem

if $\mathbf{x} \sim geometric(p)$ *then* $P(X > x + y)$

Negative Binomial distribution

Exponential- Geometric

Exponential distribution

$\mathbf{X} \sim exp(\lambda)$ count the time until a Bernoulli trial it is success, the trials are continous at a time of success λ

Count vowels

```
def count_vowels(iterable):
  vowel=0
  for i in iterable:
    if i in ['a','e','i','o','u']:
      vowel += 1
  return vowel
print(count_vowels('abcdefgjhjhmmaslasdoasjd'))
```

Flag strategy

is K a prime number?

```
def prime(k):
  i =1
  flag = False
  while flag==False and i<k-1:
    i = i+1
    if k%i==0:
      flag=True
  if flag==True:
    return 'no primo'
  else:
    return 'primo'
prime(100)
```

Matrix operations

Transpose

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, A^T = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$$

```
matrix = [[1,2,3],[10,11,12],[18,19,29]]
def transpose(matx):
  transpose_mat=[]
  for h in range(0,len(matx[0])):
    row =[]
    for j in matx:
      row.append(j[h])
    transpose_mat.append(row)
  return transpose_mat
```

Matrix operations

Identity Matrix

$$I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, A_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
def nrow(size,row_n):
    row = []
    for j in range(0,size):
      if j == row_n-1:
        row.append(1)
      else:
        row.append(0)
    return row
```

Matrix operations

Identity matrix

```
def indentity(matrix):  # we also could take min ( column, row)
    and return a square of these size
  # check if square
  check = len(matrix)
  for j in matrix:
    if len(j)!= check:
      raise Exception("Not is a square MATRIX")
  mat=[]
  for j in range(1,check+1):
    mat.append(nrow(check,j))
  return mat
```

sum of matrix

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} A + B = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{pmatrix}$$

```
def sum_matrix(matA, matB):
  rows = len(matA)
  column =len(matA[0])
  if len(matA) != len(matB):
    raise Exception('Dont have the same number of rows') # using
        transpose we can check the number of columns!
  sum_mat = []
  for i in range(0,rows):
    row =[]
    for j in range(0,column):
      row.append(matA[i][j]+matB[i][j])
    sum_mat.append(row)
  return sum_mat
```

Matrix multiplication

The inner product defined as $\vec{x}.\vec{y} = \sum x_i y_i$

```
def inner(vectorA,vectorB):
  if len(vectorA) != len(vectorB):
    raise Exception('overcome dimensionality')
  column = len(vectorA)
  addedVector = []
  for j in range(0,column):
    if type(vectorB[j])==list:
      addedVector.append(vectorA[j] * vectorB[j][0])
    if type(vectorB[j])!=list:
      addedVector.append(vectorA[j] * vectorB[j])
  return sum(addedVector)
```

Matrix multiplication

$$AB = C = [c_{ij}] \tag{15}$$

$$c_{ij} = A_i.B_j$$

thus c_{ij} is the inner product of the $i - row$ of matrix A and the $j - column$ of matrix B.

```
def prod(matA, matB):
 row = len(matA)
 col = len(transpose(matB))
 resultado = []
 for j in range(0,row):
   row =[]
   for i in range(0,col):
    row.append(inner(matA[j],transpose(matB)[i]))
   resultado.append(row)
 return resultado
```

Dynamic code

```
for x in range(3):
    exec(f'var_{x}=x')
```

Most popular libraries by statistical analysis

- pandas
- matplotlib
- seaborn
- os
- statsmodels
- scipy
- numpy

let's star to programming

you must type all that appear in the following theme, is very useful uses the script.

```
import stats
print(str(python for all))
```

STATISTICS library

statistics library is integrated by default with python installation it is useful to calculate, mean, median , mode and other statistics.

```
import statistics
data=[1,2,3,4,5]
statistics.mean(data) #arithmetic mean
```

statistics functions

1. mean()
2. median()
3. mode()
4. stdev()
5. variance()

Working directory

the working directory is a path that indicates a python where input and otput files for instance datasets, images or documents. to know by default what is the current workign directory:

```
import os
print(os.getcwd())
dir=os.getcwd()
print(dir)
```

Normality

The normality is a concept derived of nature, that was mathematically encripted in some symbols:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(x-\mu)^2}{2\sigma^2}} \tag{16}$$

Histogram

what is the distribution of the variable regarding it self?

```
plt.hist()
```

Visual relation between two variables

scatter

```
plt.scatter(x,y,data="dataname")
```

we can compare strings. indexing strings. what it is the last position.

`element[-1],`

Remember that in python index start in 0 and end in n-1.

```
lista[beging:stop:step]
lista[::] = lista[0:len(s):1]
```

Break statement

Sometimes we need stop the run or execution of a program, the **break**

```
k=222
limit = 19
ls=[]
for j in range(k):
    if j%2==0 :
        ls.append(j)
    if len(ls)>limit:
        break
print(ls)
print(len(ls))
```

The previous code will finish when the length of the list, reach a size greater than the limit.

Continue statement

In some cases we need avoid a condition, remaining in the cycle.

```
N=10
ls=[]
for l in range(N):
    if l==4 or l==6:
        continue
    elif l%2==0:
        ls.append(l)
print(ls)
```

Sorting

```
dat=[99,23,13,44,120,42,12]
new=[]
while len(dat)!=0 :
    k=dat[0]
    for j in dat:
        if k < j:
            k=j
    dat.remove(k)
    new.append(k)
print(new)
```

Insights about recursion

The process must be finish, in some cases, therefore we need establish a **base case**, that means the instance of the problem in wich the recursion must be over.

Pandas

It is open source library

Nature and normality

Pandas

```
categorial = []
nonormal = []
normal = []
for t in df.columns:
    if df[t].dtypes=="object":
        categorial.append(t)
    if df[t].dtypes=="int64" or df[t].dtypes=="float64":
            n,p = stats.shapiro(df[t])
            if p<0.05:
                nonormal.append(t)
            else:
                normal.append(t)
```

Removing missing values

Pandas

This code will remove the missing values in all columns or rows.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data = pd.DataFrame([[2,10,np.nan], [30,60,np.nan],[
    np.nan,np.nan,np.nan]])
data.dropna(axis=0, how='all', inplace=True)
data.dropna(axis=1, how='all', inplace=True)
```

good words: unsavory: desagradable rushed:precipitarse, apurado.
arguibly: posiblemente amaze; asombrar. advertise: anunciar

Replacement -Order doest not matter

Poisson Distribution

Poisson distribution is derived from when the variable follow a binomial distribution with a $n \to \infty$

In the limit case, the occurrence of a only event is only guaranteed in the measure that the space is very small, for instance if the ocurrence of the events is simultaneous, you should not consider a Poisson distribution.

$E(x) = np = \lambda$, thus ,$\frac{\lambda}{n} = p$ and according to FD of a $x \sim b(n, p)$

$$\begin{aligned} PDF =& \frac{n!}{(n-k)!k!}(\frac{\lambda}{n})^k(1-\frac{\lambda}{n})^{n-k} \\ =& \frac{(n-k+1)!}{n^k k!}(1-\frac{\lambda}{n})^n(1-\frac{\lambda}{n})^{-k} \end{aligned} \tag{17}$$

$e = \lim_{x\to\infty}(1+\frac{1}{n})^n$we must use $t = \frac{n}{k}$, and thus $\frac{n+k}{n} = 1 + \frac{k}{n}$

$\lim_{n\to\infty} = \frac{e^{-k}\lambda^k}{k!}$

$$P(X = x) = \frac{e^{-x}\lambda^x}{x!} \tag{18}$$

Conjoint probability

fooling;engañado. disguise: crew:tripulación,personal. contestant: spoil:arruinar, estropear. try it out: scathing honing:afilado. boggling: endeavour: esfuerzo, empeño. wrecked: destrozado, arruinar. bestowed: otorgado. dumped: claim: afirmar. fully: blunder: conceals: oculta. disregard: ignorar.

inclusion-exclusion

Independece

Independence not is equal not occurrence, the independence is related with the change of the probability that occur a event given that another occur.

$$P(A \cap B) = P(A)P(B) \tag{19}$$

Tossing coins it is a bernoulli trial, and the occurrence of a event not affect the another event.

Contingency table

		Diagnose	
		Disease	No-Disease
Risk Factor	Smoke	a	b
	Not Smoke	c	d

What it is $P(Disase \mid Smoke) = \frac{a}{a+b}$ note that marginal distribution. Note that $P(Disease \cap Smoke) = \frac{a}{(a+b+c+d)}$. Also note that $P(Smoke) = \frac{a+b}{(a+b+c+d)}$. Note the result of divide the last two probabilities.

Conditional Probability

The probability of event given a "information". *Probability of A occur given B occurs.*

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \tag{20}$$

Note that $P(A \cap B)$ it is equal to $P(B \cap A)$.

$$P(A \mid B)P(B) = P(B \mid A)P(A) \tag{21}$$

Bayes theorem

Notice that not is same: *The probability that occur A given B, that occur B given A.* However we can compute one of the another.

$$P(A \mid B) = \frac{P(A)P(B \mid A)}{P(B)} \tag{22}$$

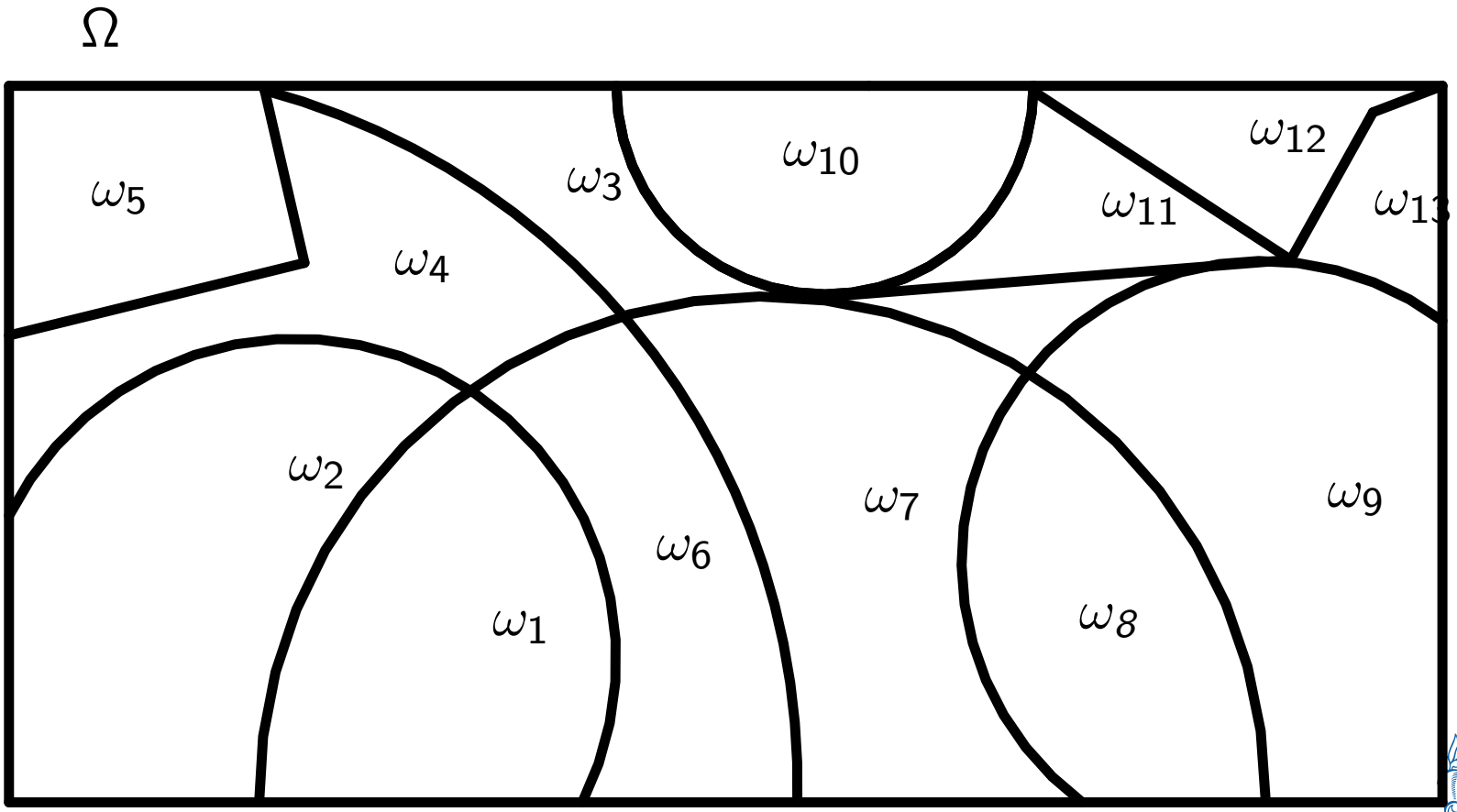
Bowls problems

Derive the following problem.

Law of total probability

The sample space defined as Ω if we split omega in ω k subsets in order that each *subset* no overlap with others. $\bigcup_{i=1}^{k} \omega_i = \Omega$ y $\bigcap_{i=1}^{k} \omega_i = \emptyset$ For instance the sample space defined as $\Omega = \{a, b, c, d, e, f\}$ $\omega_1 = \{a, f\}$ $\omega_2 = \{b, c, d\}$ $\omega_3 = \{e\}$.

Split Ω


Ω
ω_5
ω_4
ω_3
ω_{10}
ω_{11}
ω_{12}
ω_{13}
ω_2
ω_1
ω_6
ω_7
ω_8
ω_9

$P(A)$

total law probability

We need remember by set theory that a event A could be rewrite as $A = (A \cap B) \cup (A \cap C)$. if $(B \cup C) = \Omega$ for this case we can rewrite

$$\begin{aligned} A &= (A \cap \omega_1) \cup (A \cap \omega_2) ... (A \cap \omega_k) \\ P(A) &= P(A \cap \omega_1) + ... + P(A \cap \omega_k) \\ P(A) &= P(A \mid \omega_1)P(\omega_1) + P(A \mid \omega_2)P(\omega_2) + ... + P(A \mid \omega_k)P(\omega_k) \end{aligned} \tag{23}$$

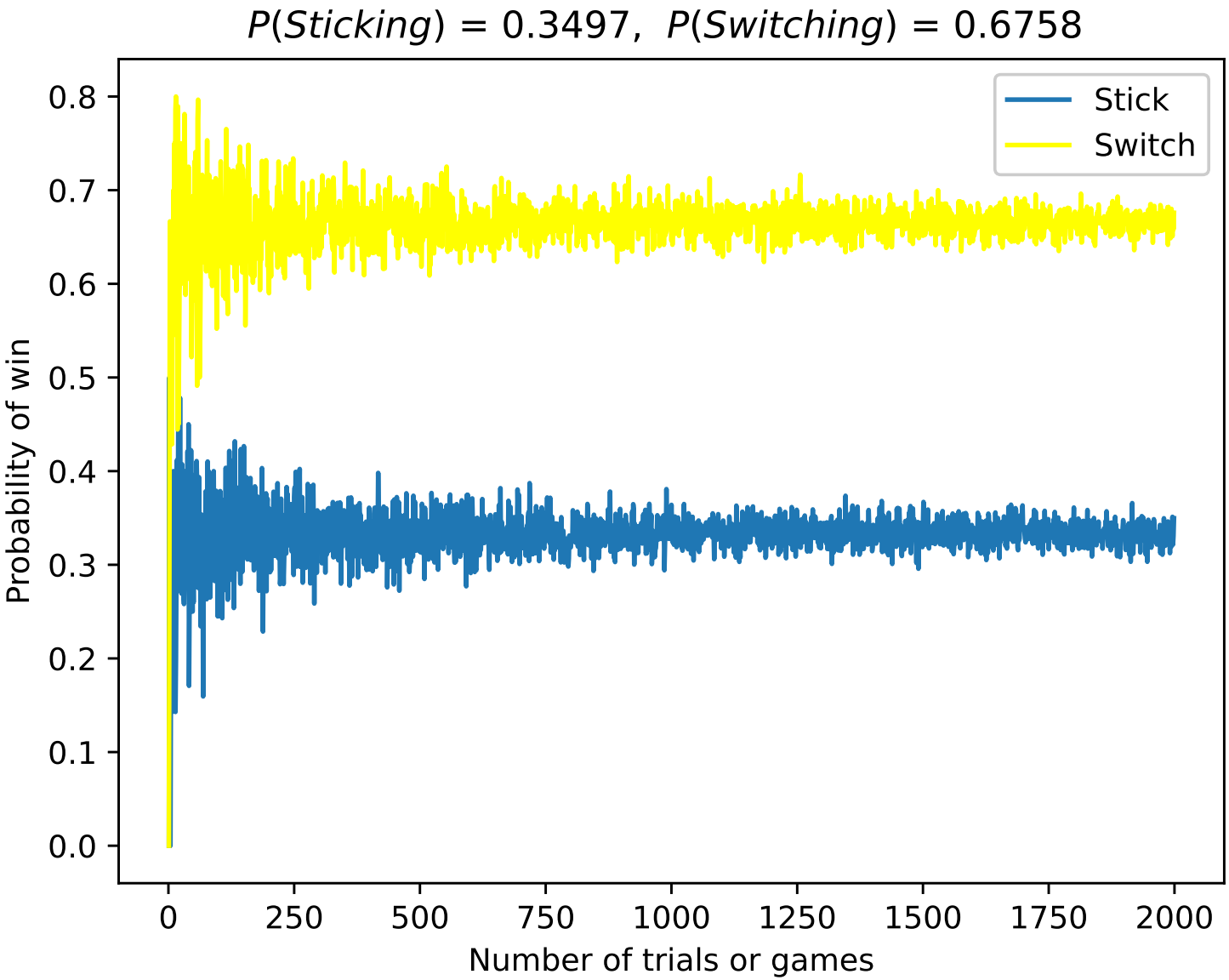
note that by total law $P(A)$

Monty Hall simulation

This problem is so not intuitive that generated controversy in the academic community.

There are three closed doors, and you must select one to win a car, behind the doors, there is a car, and behind the other two there are goats. After select the door, another door with a goat it is showed to you, then you can change your first election, the main problem is if you must remain in the selected door or change?
Notice that the initial probability of win the car is the 1/3.

Monty Hall simulation



Monty Hall problem

Python code

```
def monty_game():
  doors=[1,2,3]
  doors_variable = doors.copy()
  winner = random.randint(1,3)
  select_one = random.randint(1,3)
  values = [winner,select_one]
  switch = list( set(doors) - set(values))
  select_two = random.randint(switch[0], switch[-1])
  s1,s2 = 0,0
  if winner == select_one:
    s1 += 1
  else:
    s2 +=1
  return [s1,s2]
```

Monty Hall problem

Python code

```
trials=2000
s1_=[]
s2_=[]
for n in range(1,trials):
  prob_s1 = sum([monty_game()[0] for _ in range(1,n)])/n
  s1_.append(prob_s1)
  prob_s2 = sum([monty_game()[1] for _ in range(1,n)])/n
  s2_.append(prob_s2)
plt.plot(np.arange(1,trials),s1_, label='Stick')
plt.plot(np.arange(1,trials),s2_, label='Switch',color='yellow')
plt.title('$P(switching)$ = {:.4}, $P(sticking)$ = {:.4}
    '.format(prob_s1,prob_s2))
```

Monty Hall

Bayesian solution

The event D_i the i winner door and M_j monty open j door, for $i, j = 1, 2, 3$.

$$P(D_i \mid M_j) = \frac{P(M_j \mid D_i)P(D_i)}{P(M_j)} \tag{24}$$

you select the first door and monty the second, therefore the question is $P(D_3 \mid M_2)$. Notice that $P(M_2) = \sum_{i=1}^{3} P(M_2 \mid D_i)$. given the rules of game, $P(M_2 \mid D_2) = 0$, and $P(M_2 \mid D_3) = 1$, if monty could select random in two choices $P(M_2 \mid C_1) = 1/2$ and finally, switch strategy have a probability of $2/3$.

Deck of cards

52 standard

There are 4 'types'(suits), clubs, diamonds,

Gamblers ruin

Topics to research

- Conditional indepedence
- Panzer ruin
- vander

Buffon Needled

Here we have a column

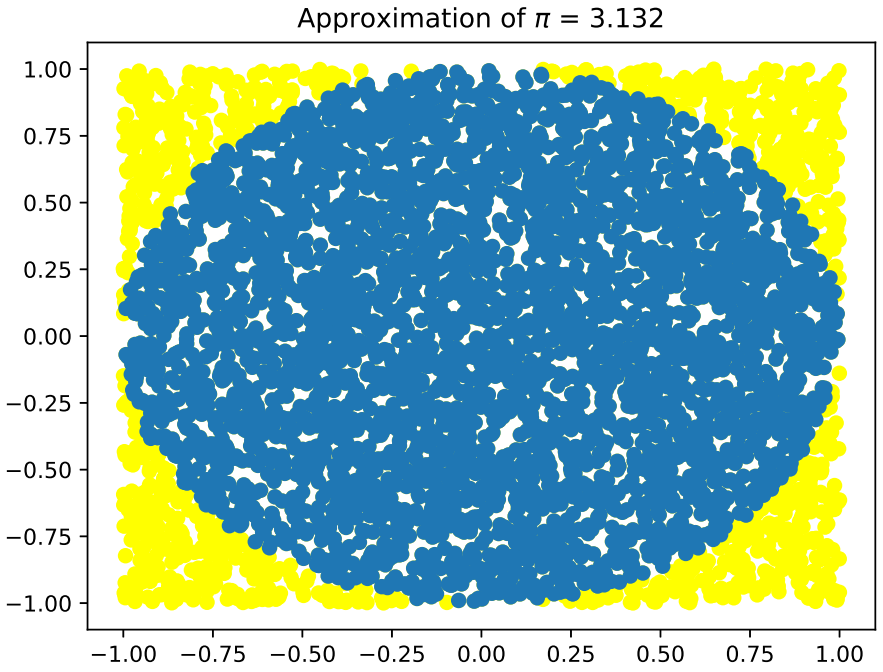
we have another column

π Number

```
import numpy as np
import matplotlib.pyplot as plt
dots = 5000
c1,c2=-1,1
x = np.random.uniform(c1,c2,
    size=dots)
y = np.random.uniform(c1,c2,
    size=dots)
coordenates_circle =
    (x**2)+(y**2) < 1
circle_y=y[coordenates_circle]
circle_x=x[coordenates_circle]
pi = 4*sum(coordenates_circle)
    / dots
plt.scatter(x,y, color='yellow')
plt.scatter(circle_x,circle_y)
```

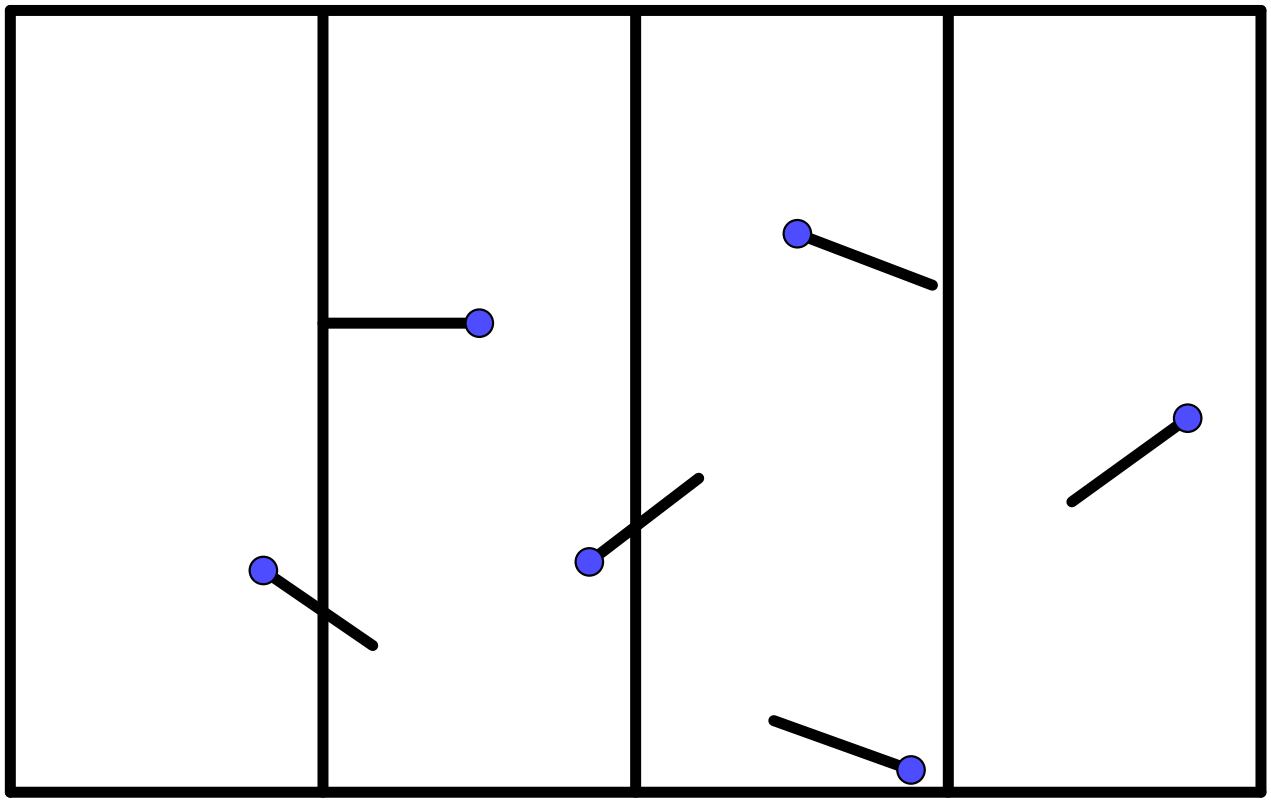
What is the probability of a drop lands in the circle?

$$P(hit) = \frac{\pi r^2}{4r^2} \tag{25}$$



Needles

Proposed by and resolved by the naturalist. take in mind the



Uniform distribution

$\mathbf{x} \sim U(a, b)$ in the interval(a, b).

$$f(x) = \frac{1}{b-a} \tag{26}$$

the function is defined in the open interval $a < x < b$. Remember that:

$$F(x) = \int_{-\infty}^{x} f(u)du \tag{27}$$

```
import numpy as np
np.random.uniform(a,b
    ,size=(k,p)) # []

#Draw k list with p elements
    with numbers [a,b)
```

Choose a point in the interval (a,b), you can calculate what it is the probability that a point is in (c,d)

Project

Collecting to Centroid

Write a program to allow to the user find the points closest to a k centroids.

Project

```
import numpy as np
info = {'key_one':'1,2,3', 'key_two':'2'}
info['key_one']
lista = info['key_one'].split(',')
lista = [int(x) for x in lista]
sum(lista)
```

Numpy

The uses of pure python structures could not be efficient due are unwieldy. Because the language not was devised for numerical computations. It is written in C. Related to Cpython interpreter.

Module

In your current directory you can save the following code:

```
def fact(n):
    if n==1:
        return 1
    else:
        return n*fact(n-1)
```

and save the file as **fact_recur.py** enter to the shell and type:

```
python3
import fact_recur
```

The function not be loaded automatically, we need access to dot notation with the module name.

delete a function

```
del funcition_Name
```

Module

You can assing a function to a local name

```
import fact_recur
factorial = fact_recur.fact
```

Modules and packages

import it is a keyword the package will be load if is in the path. Python search the module in the current directory, and after in PYTHONPATH

```
sys.path #variable that contain names of directories to load
    modules.
```

Environment variables

Type in **shell**

```
echo $LANG
echo $PATH
echo $USER
echo $HOME
```

For instance the environment variable **PATH** it is for searching executable files. We can create a variable typing in shell

```
VARTEXT='Text_in_example'
```

to see the variable:

```
echo $VARTEXT
```

local variables are temporary variables.

tips to write in English

- as your program gets longer
- boils down; se reduce.
- We could have also done

Namespace

Could exist one o more definition of the same variable name, living in different namespace. The **scope** of a variable, **global** variable could be invoked inside or outside of a function, while if you define a variable inside a function means that this is **local** variable, or this could not be invoked outside of the function where it was defined.

local and global scope

global refer to the ability to access in all program, and **local** only for parts of code. when you define a variable or assign a value inside a function the scope of the variable is local.
Thus if two variables have the same name, a local variable override the global variable.

Namespace and scope

Variables are identifiers mapping to the objects stored in memory. A Dictionary of variable names is called **Namespace**. Each function has its corresponding namespace.

scope

```
x='old_value'
def change_x():
  x = 'new_value'
  print(x)
change_x()
print(x)
```

this will return:

```
old
```

```
'x' in globals()
```

Error

You can not modify a global variable inside a function. Unless you use the **global** keyword.

```
x = 'old_value'
def increase_word():
  global x
  x ='new_value'
  print(x)
increase_word()
print(x)
```

this will return:

```
new_value
```

Notice that we can create a global scope outside of global space.

π as global

Monte carlo simulation

```
import numpy as np
def Pi(dots):
    global pi
    c1,c2=-1,1
    x = np.random.uniform(c1,c2, size=dots)
    y = np.random.uniform(c1,c2, size=dots)
    coordenates_circle = (x**2)+(y**2) < 1
    pi= 4*sum(coordenates_circle) / dots
```

You can check that *pi* is a global variable using **globals()** function.
What is the relationship with the keyword **return**?.

LEGB rule

Local, Enclosing, Global, and Built-in

Paradigm of scope with this avoid collisions by names.
python names could came from ; Assigments, import, def, and class.

- local scope: in python functions or lambda expressions.
- enclosing or nolocal: nested functions.
- global: related to module, visible in all your program.
- built-in: this have keywords, functions, exceptions, that are built in. This hierarchical structure or pattern of searching to call a variable.

Remember that are function definition and function call.

scope is related with namespace "python scope are implemented as dictionary mapping identifiers 'name variables' to values or objects. " these dictionaries are called namespaces"

"In each function call is created a scope. the local scope is destroyed when the function return."

```
def functionOne(var):
    result = ....
def functionTwo(var):
    result= ....
```

Notice tat **var** and **result** dont clash due both are defined in a local scope and could no be invoked from another side of its one scope.

Symbol table

A data structure that contain information about, method, classes, variables and so on of a program.

- global symbol table
- local symbol table

```
globals()
locals()
```

When you import a module this not is loaded automatically in symbol table only the module name, therefore is needed access by the module name. It is important to know that each module have its own symbol table.

dir

the **dir()** function tell us what things are inside of one package.

```
dir(module_name)
```

Could be useful for do not forget methods or attributes inside of package.

__main__

Sometimes you write a module to be imported and its behavior must be change in comparation with itself is the main program to be executed or be part of another program.

```
if __name__ == 'main':
    print('The main program never will be imported')
else:
    print('Was loaded a not main program')
```

save this snippet with **two.py**

```
import two
print(' This is main and was be executed from shell')
```

save as **one.py** and type in shell

```
python3 one.py
```

__main__

When the program no it is the main or module is called from another module then **__name__** take the name of module. This allow us avoid, execute part of the code when it is imported for instance to don show a graph in a imported module.

Zip

Loop

zip gather elements and return a tuple with each k corresponding elements in x iterable objects.

```
a= [1,2,3,4]
b= ['red', 'blue','black', 'white']
pairs = list(zip(a,b))
```

decorators

os library

We used previously this library to change the working directory, however it has another important extensions thus:

```
os.remove() # Remove a file
os.rmdir() # Remove a directory
os.rename() # Rename a file
```

sys library

```
import sys
lista = sys.arg
print(lista)
```

will return a list with the arguments to executed in the shell.

```
python3 module.py
```

```
sys.platform # Return the os of machine.
sys.path # where modules will be loaded.
sys.path.append('/home/ces/modulos/python')
```

π from shell

Save a module called pi.py

```
import sys
import numpy as np
def Pi(dots):
    c1,c2=-1,1
    x = np.random.uniform(c1,c2, size=dots)
    y = np.random.uniform(c1,c2, size=dots)
    coordenates_circle = (x**2)+(y**2) < 1
    pi= 4*sum(coordenates_circle) / dots
    return pi
print(Pi(int(sys.argv[1])))
```

```
python3 pi.py 100000
```

sys

you can also change the prompts in shell

```
import sys
sys.ps1='..>'
sys.ps2='missing...'
```

Information about a function

```
Pi.__code__.co_varnames
Pi.__code__.co_consts
```

init

The __init__.py files are required to make Python treat directories containing the file as packages.

Inner functions

```
def outer():
    statements
    varout = value
    def inner():
        statements
```

The **outer** or **enclosing** the outer function share to the inner function its namespace. For instance inner could be call **varout** variable.

Pi information

```
import numpy as np
def Pi(dots):
    c1,c2=-1,1
    x = np.random.uniform(c1,c2, size=dots)
    y = np.random.uniform(c1,c2, size=dots)
    coordenates_circle = (x**2)+(y**2) < 1
    pi= 4*sum(coordenates_circle) / dots

    def IPi():
        print(f'pi was roughly {pi} with {dots} iterations')
    IPi()
```

dots and **pi** are **nonlocal names** for **IPi()** function.

inner functions

Provide encapsulation

A nested or inner function could not be invoked outside of the closing function.

topics to research

- isinstance
- raise
-


IHS
Pontificia Universidad
JAVERIANA
Bogotá