Introduction a first course in Programming to Data Science

Using python.

Iván Andrés Trujilllo Abella

Facultad de Ingenieria
Pontificia Universidad Javeriana

trujilloiv@javeriana.edu.co

What is involved in programming

- creativity
- abstraction
- divisibility

Algorithmic thinking

This is more important that a concrete language.

Logical puzzle

two tribes one said always True and other always said False (note that there are boolean not change).

when you want print a 'quote' inside a text uses backslash

```
print('it\'s mine')
```

when there are a 'enter' uses triple quotes.

key words

a identifier ('variable name') avoid uses words as

```
def
class
elif
list
while
...
```

Arithmetic operators

could be binary or unary and each one have a level of hierarchy.

```
2**4 # Exponentiation
4 //2 # integer division
3%2 # Module
```

```
2**4+1
(2**4)+1
```

the exponentiation have the major hierarchy then will be executed first.

Operator precedence

to the left to the right major hierarchy (),**, *, / , +-.

```
a = 2
b = 4
c = 9
d =  5
d+a**c/b-d
```

Which is the value?, we can calculate it step by step:

```
r = 2**2
r = r/4
r = r + 5
r = r -9
print(r)
```

When two operators have the same precedence then is evaluated the left one first, in exponential operator is evaluated the right one first.

data types

Boolean

True or False

Boolean rules

boolean rules of the conjuction and disjunction and negative (logical operators). Internally True is (1) and False (0).

```
and
or
not
```

- x and True == x
- x or False == x
- x or True == True
- x and False == False
- x and x == x
- x or x == x

conditionals

asses a condition return a boolean value, then the program will execute a block if the value is True other wise not execute the block

```
if condition:
    #Block
    statements
```

```
x = 3
if x > 2 :
    print('x-is a number grater than 2')
```

Conditionals

when appear else will be executed the else block if the condition is False.

```
if condition:
    # if block
    statements
else:
    # else-block
    statements
```

conditionals

elif

in some cases we need asses one more of a condition for instance if the variable is in different ranges then we need uses elif:

```
if condition:
    #if - block
elif condition:
    # elif - block:
elif condition:
    # elif - block:
...
else:
    # else - block
```

Why is better uses elif instead if? , the answer is the reduction of assessments, note that if a elif expression is True then the program skip to get out of else, nevertheless if we replace elif by if then all if conditions will be checked. is valid also the program without else.

Switch

Some programs have switch python could be recursively uses one method.

is, is not

Difference between is and ==

loops

while

repeat code until reach a condition, for instance we need print in screen the number from 0 to 10.

```
i = 0
while i < 11:
    print(i)
    i  = i + 1
```

Note here that the statements **print(i)** and $\mathbf{i = i + 1}$ will be executed until $\mathbf{i < 11}$ be **False**. Remember that python first assess the right hand and after assign to **i**.

loops

for

It is a short hand of while loop.

```
for i in range(0,11):
    print(i)
```

Note here that we not initialize the variable **i=0** and also do not define a exit statement $\mathbf{i = i + 1}$ Note that range create a sequence that begin in 0 and end in $n - 1$.

for vs while

```
# While implementation
i = 0
while i < 10:
  k = 0
  while k < 10 :
    j = 0
    while j < 10 :
      print(i,k,j)
      j = j + 1
    k = k + 1
  i = i +1
#--------------------#
# for implementation
for x in range(10):
    for y in range(10):
        for z in range(10):
            print(x,y,z)
```

Example flag

```
def searchk(k,l):
  flag = False
  counter = 0
  for x in l:
    if x == k :
      flag = True
    counter += 1
  return flag,counter
```

Example flag

```
def searchop(k,ls):
  i = 0
  counter = 0
  flag = False
  while i < len(lisn) and flag==False:
    if lisn[i] == k:
      flag = True
    counter +=1
    i +=1
  return flag, counter
```

Basic structures

- list
- dictionary
- tuples

Remember that list not keep the objects itself otherwise, keep the direction of memory (therefore it is neccesary uses copy method). The simple invocation:

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

aware

You can not use a variable without assignment a value, this a common mistake even in intermediate programmers.

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.

How to write a program?

- Write in a blank paper the general structure
- Probe to hand, step by step the solution
- Generalize the program
- Uses functions
- Parameters outside

Tangent line

Equation of line

$$y = \beta_0 + \beta_1 * x \tag{1}$$

Remember that β_1 is denominated as slope, or rate of change.
$f(k) = y$ could be a production function, where k is the level of capital, therefore the level of productions is determined by k.

Derivates

Calculus invented by Newton and Leibniz independently, has important applications, why is important? think in the study of the change of a variable regarding another. However the rate of change not always is a constant, for instance think the exponential growth of the population. The main idea is find the tangent line in a point of the function that describe the objective function.

Rate of change

Linear equation

$$
\begin{aligned}
f(x) &= \beta_0 + \beta_1 x \\
\frac{d}{dx} f(x) &= \beta_1
\end{aligned}
\tag{2}
$$

where this come from?, the limit definition is the guideline.

$$
f^1(x) = \frac{d}{dx} f(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{3}
$$

Applying limit definition

$$\frac{d}{dx}f(x) = \frac{\beta_0 + \beta_1(x + \Delta x) - \beta_0 + \beta_1 x}{\Delta x} \tag{4}$$
$$f^1 = \frac{\beta_1 \Delta x}{\Delta x} = \beta_1$$

In terms of the production function whats mean β_1. why the derivate of a constant function is equal to zero?

tangent of line

x^2

think in $f(x) = x^2$, the derivate of this equation is $f^1(x) = 2x$, where its come from?, the tangent line in the point x_0 is:

$$l = f(x_0) + f^1(x_0) * (x - x_0) \tag{5}$$

Python implementation

```
import matplotlib.pyplot as plt
import numpy as np
def fx(x):
  return x**2
def Dfx(x):
  return 2*x
def LDfx(x,point):
  return fx(point) + Dfx(point)* (x-point)
x = np.linspace(-20,20,1000)
point = 4
plt.plot(x, fx(x))
plt.plot(x, LDfx(x,point))
plt.scatter(point, fx(point), color='green')
```

Quadratic equation

Analytical solution

Remember a value is root of a function if $f(x_0) = 0$, the quadratic function have the following general formula $ax^2 + bx + c = 0$. We need find the root, therefore:

$$x^2 + \frac{b}{a}x + \frac{c}{a} = 0 \tag{6}$$

note that we can add a term to complete the square we need meet $2xy = \frac{b}{a}x$ thus $y = \frac{b}{2a}$. Adding to both sides, we have:

$$(x + \frac{b}{2a})^2 = (\frac{b}{2a})^2 - \frac{c}{a} \tag{7}$$

getting the solution as:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{8}$$

Analytical solution

```
import numpy as np
import matplotlib.pyplot as plt
def quadratic(a,b,c, x): #Quadratic equation
  return a*x**2 + b*x + c
def rootsQuadratic(a,b,c): #Analytical solution
  rootP = (-b-(b**2 - 4*a*c)**(1/2)) / (2*a)
  rootN = (-b+(b**2 - 4*a*c)**(1/2)) / (2*a)
  return rootP, rootN
## Testing plt.axv(h)line(x(h)=0, color='black')
a,b,c, xinf, xsup = 1,3,-22, -10, 10
n = abs(xinf) + abs(xsup) + 1
domain = np.linspace(xinf,xsup,n)
y = [quadratic(a,b,c,x) for x in domain]
plt.plot(domain,y)
rootp,rootn = rootsQuadratic(a,b,c)
```

Quadratic equation iterative

```
def quadratic(a,b,c, x):
      return a*x**2 + b*x + c
def dfQuadratic(a,b,x):
  return a*x**2 + b*x
def raphsonQuadratic(a,b,c,x0, error_max=0.0000015,
    iteration_max=100):
  xi = x0
  iter, error = 0, 100
  data = []
  while (iter < iteration_max) and (error > error_max):
    xj = xi - quadratic(a,b,c,xi) / dfQuadratic(a,b,xi)
    error = abs(xj - xi)
    iter += 1
    data.append((xj,xi,error,iter))
    xi = xj
  return data
raphsonQuadratic(a,b,c,4)[-1]
```

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. □

```
def BinarySearch(k,order_lista):
  counter = 0
  bottom = 0
  upper = len(order_lista)-1
  flag = False
  while upper >= bottom:
    counter += 1
    half = int((bottom + upper)/ 2)
    print(bottom, upper, half)
    if k == order_lista[half]:
      flag = True
    elif k > order_lista[half]:
      bottom = half + 1
    elif k < order_lista[half]:
      upper = half - 1
    return flag, counter
```

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

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

When uses a for loop we iterate over the *key_value*.

```
info = {'key_value_one:'1', 'Key_value_two:'2'}

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

the last code will be return:

```
Key_value_one
Key_value_two
```

Iterable object

Dictionary

We can access to the data associated with each *key_value*.

```
for key in info:
    print(info[key])
```

allow us to see the data stored in the array.

```
dco = {'key_one':'triangle,square,circle',
'key_two':['triangle','square','circle']}
```

Note that the dictionary allow us keep differente data structures or data types.

Dictionary

Take in mind that python it is based upon object oriented programming, we need take in mind that daya types is returned or we are using

```
type(object) # return data type
```

and therefore we can add to *key_two* a new element to the value associated in this key.

```
dco['Key_two'].append('rectangle')
```

Note that we uses **append()** that is a method of lists.

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)
```

Example 2

ordering each list inverse

```
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)
```

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)
```

signature

```
def function(parameter_one: int, parameter_two:float) -> float
```

Docstring

The function documentation is called docstring:

```
def power(x,n):
  """
  This is docstring that will be invoked
  with the function help(power)
  ---------------
  power(x,n)
  x it is a any number that will be powered to n
  ---------------
  """
  return x**n
```

you can peek using the help function

```
help(power)
```

Factorial function

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

$$\begin{aligned} n! &= n(n-1)(n-2)(n-3)...1 \\ n! &= n(n-1)! \end{aligned} \tag{9}$$

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
```

Count

```
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('abcdd'))
```

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)
```

```
def mlR(a,b):
  if b==1:
    return a
  else:
    return a + mlR(a,b-1)
```

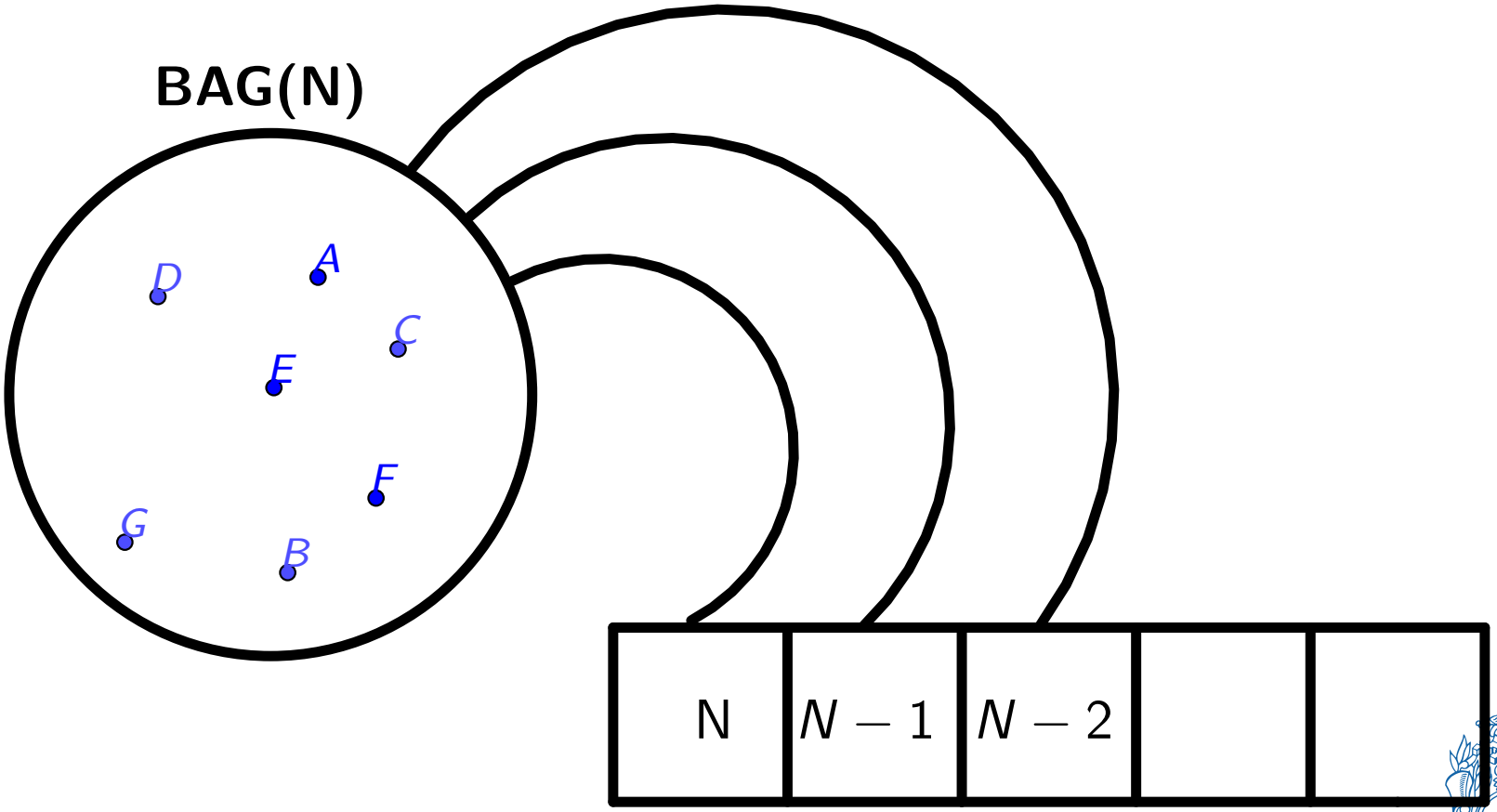
Combinatorial insights

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.

$$P_k^n = N(N-1)(N-2)(N-3)...(N-k+1) \tag{10}$$

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.

$$C_k^n = \frac{P_k^n}{k!} = \frac{n!}{(n-k!)k!} \tag{11}$$

Variation

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

$$V_k^n = n^k \tag{12}$$

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;

$$\frac{n(n-1)(n-2)(n-3)\ldots(n-r+2)a^{n-r+1}b^{r-1}}{(r-1)!} \quad (14)$$

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*

$$P(A \cup A^c) = 1 \tag{15}$$

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

$$P(A \cup B) = P(A) + P(B) \tag{16}$$

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 :

$$1 - \left(\prod_{i=311}^{360} i(360^{-50}) \right) \approx 0,97 \tag{17}$$

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 108 show the relation among the probability of two coincidences among two person in the sample.

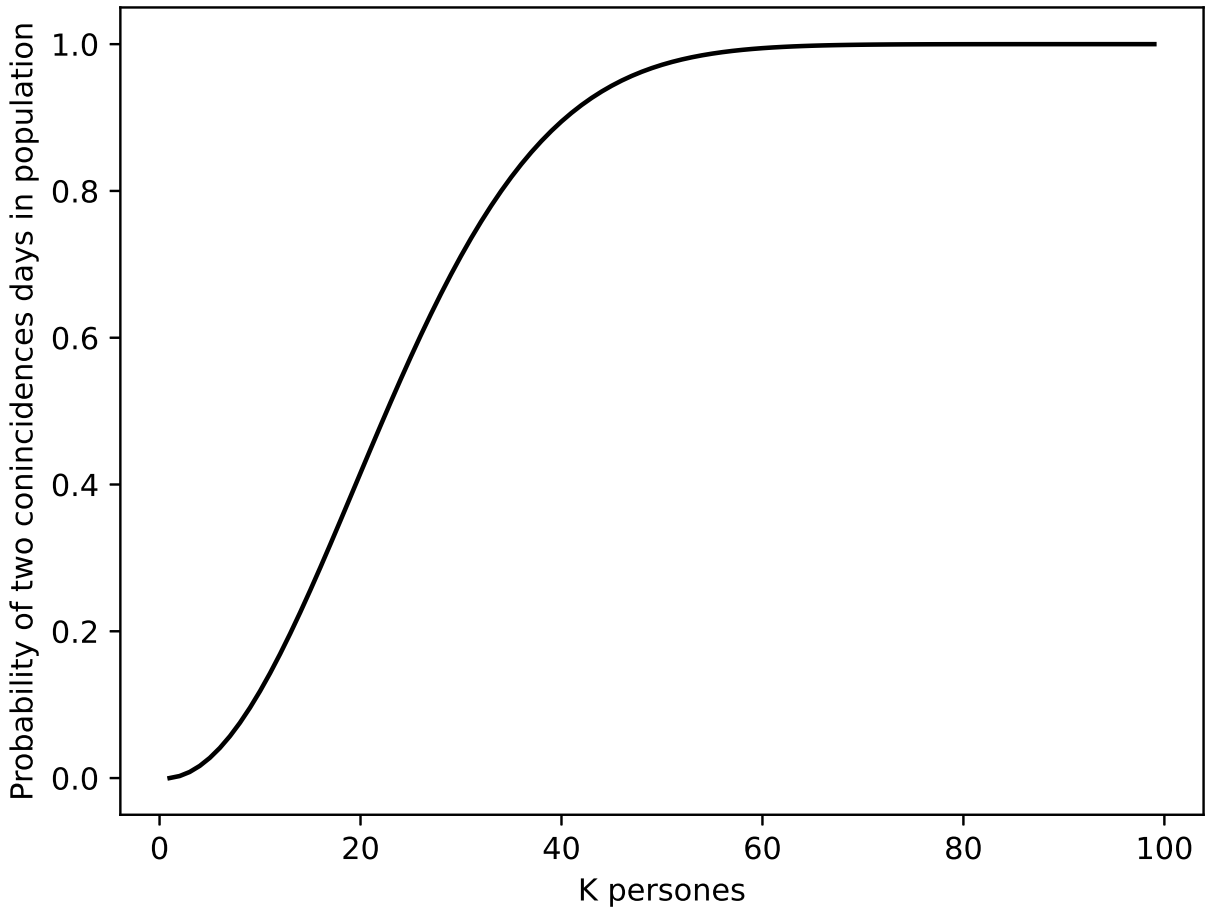


Figure: Birthday paradox with 360 days

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

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

$$\sum_{i=1}^{n} \binom{n}{i} \tag{18}$$

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

$$(x+y)^n = 1 + \sum_{k=1}^{n} \binom{n}{k} x^{n-k} y^k \tag{19}$$

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

$$2^n - 1 = \sum_{i=1}^{n} \binom{n}{k} \tag{20}$$

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.

$$P(X = x) = (1 - p)^{x-1}p \tag{21}$$

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 $P(X > x)$.

$$CDF = 1 - (1 - p)^k \tag{22}$$

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 $x \sim geometric(p)$ *then* $P(X > x + y)$

Negative Binomial distribution

Exponential- Geometric

Exponential distribution

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

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:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(x-\mu)^2}{2\sigma^2}} \tag{23}$$

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.

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 $x \sim b(n, p)$

$$\begin{aligned} PDF =& \frac{n!}{(n-k)!k!}(\frac{\lambda}{n})^k(1-\frac{\lambda}{n})^{n-k} \\ =& \frac{(n-k+1)!}{n^k k!}(1-\frac{\lambda}{n})^n(1-\frac{\lambda}{n})^{-k} \end{aligned} \tag{24}$$

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

$$P(X = x) = \frac{e^{-x}\lambda^x}{x!} \tag{25}$$

Conjoint probability

fooling;engañado. disguise: crew:tripulación,personal. contestant: spoil:arruinar, estropear. try it out: scathing honing:afilado. boggling: 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.

$$P(A \cap B) = P(A)P(B) \tag{26}$$

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.*

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \tag{27}$$

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

$$P(A \mid B)P(B) = P(B \mid A)P(A) \tag{28}$$

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.

$$P(A \mid B) = \frac{P(A)P(B \mid A)}{P(B)} \tag{29}$$

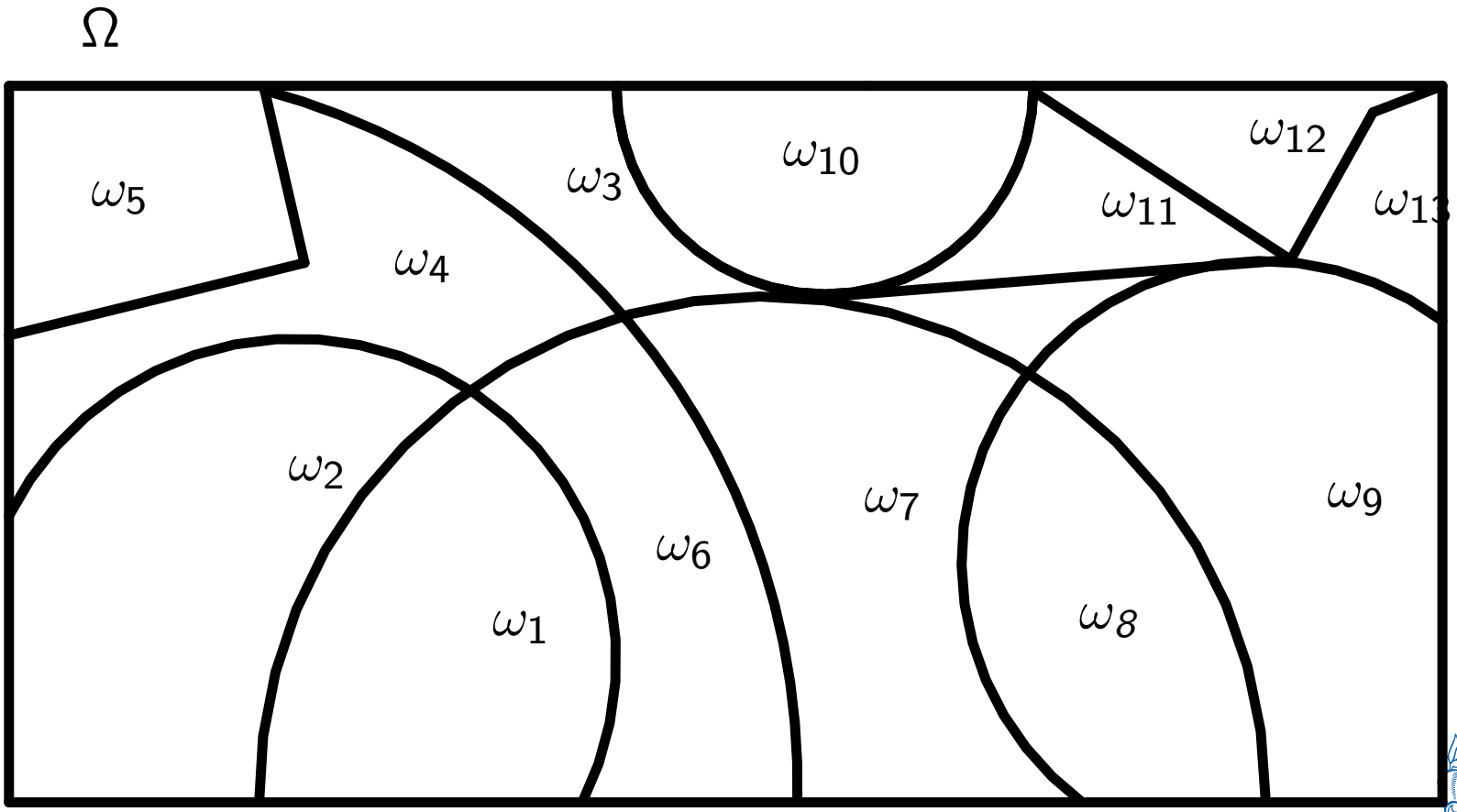
Bowls problems

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

$$\begin{aligned} A &= (A \cap \omega_1) \cup (A \cap \omega_2) ... (A \cap \omega_k) \\ P(A) &= P(A \cap \omega_1) + ... + P(A \cap \omega_k) \\ P(A) &= P(A \mid \omega_1)P(\omega_1) + P(A \mid \omega_2)P(\omega_2) + ... + P(A \mid \omega_k)P(\omega_k) \end{aligned} \tag{30}$$

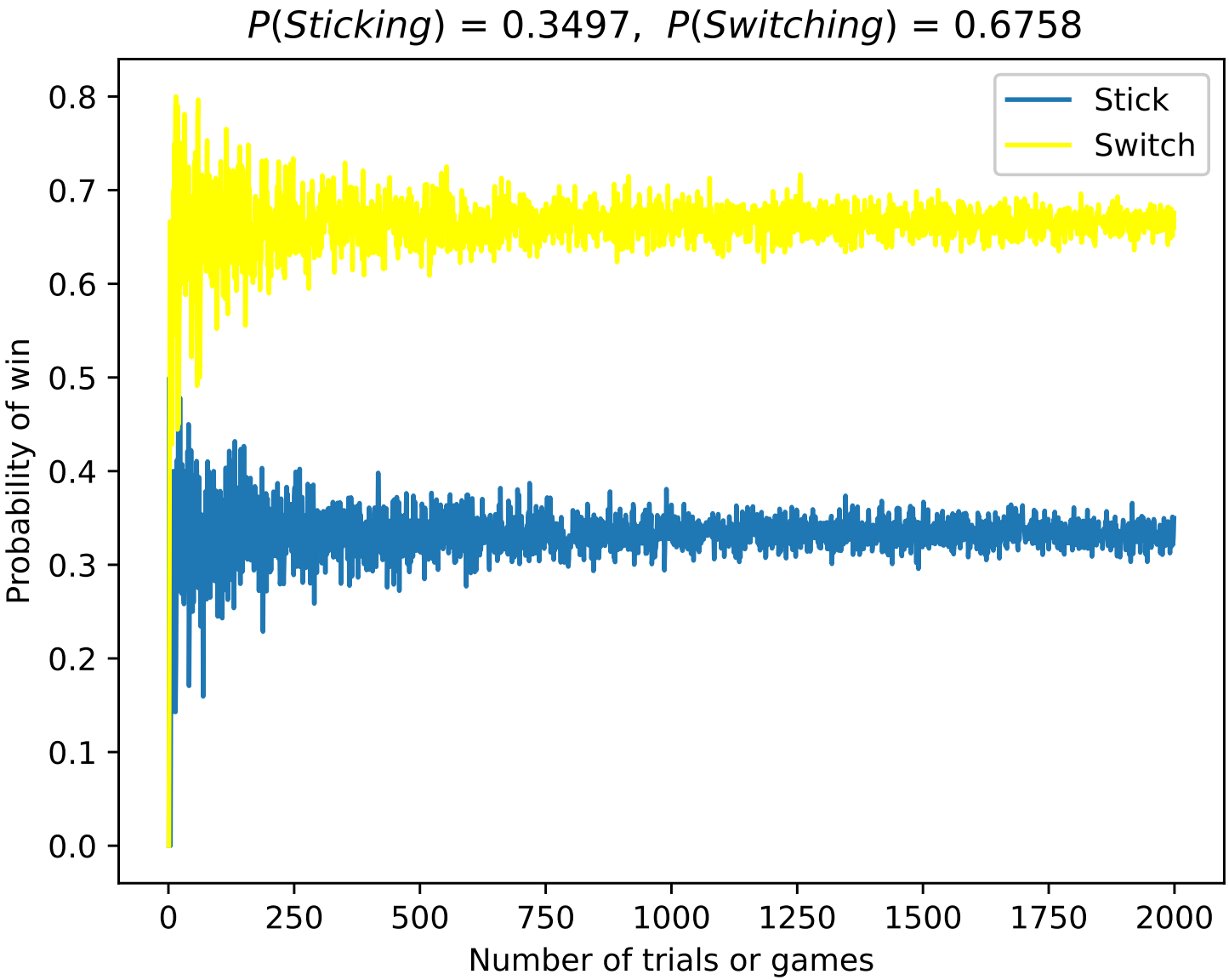
note that by total law $P(A)$

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 $i, j = 1, 2, 3$.

$$P(D_i \mid M_j) = \frac{P(M_j \mid D_i)P(D_i)}{P(M_j)} \tag{31}$$

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 $2/3$.

Buffon Needled

Here we have a column

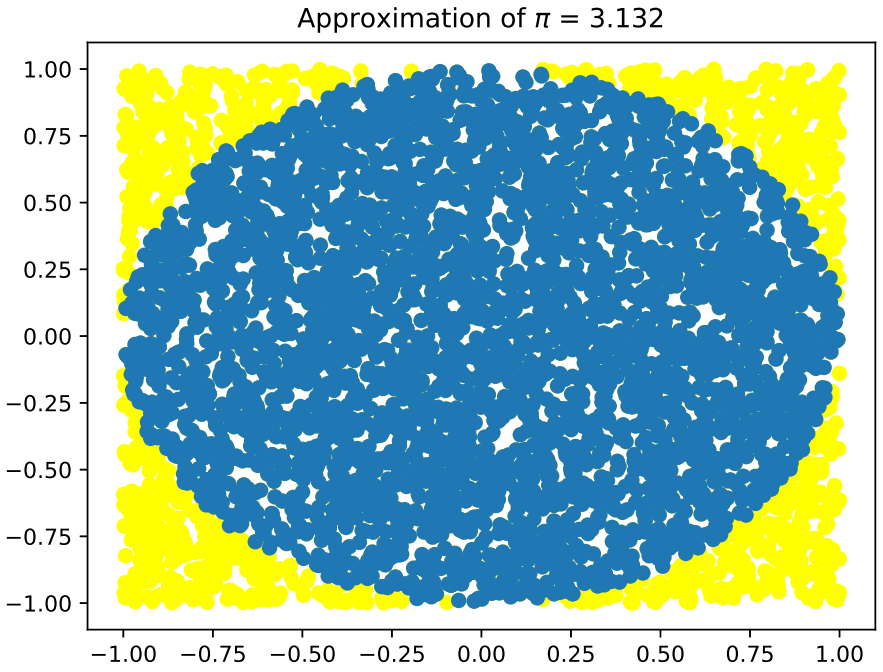
we have another column

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)
```

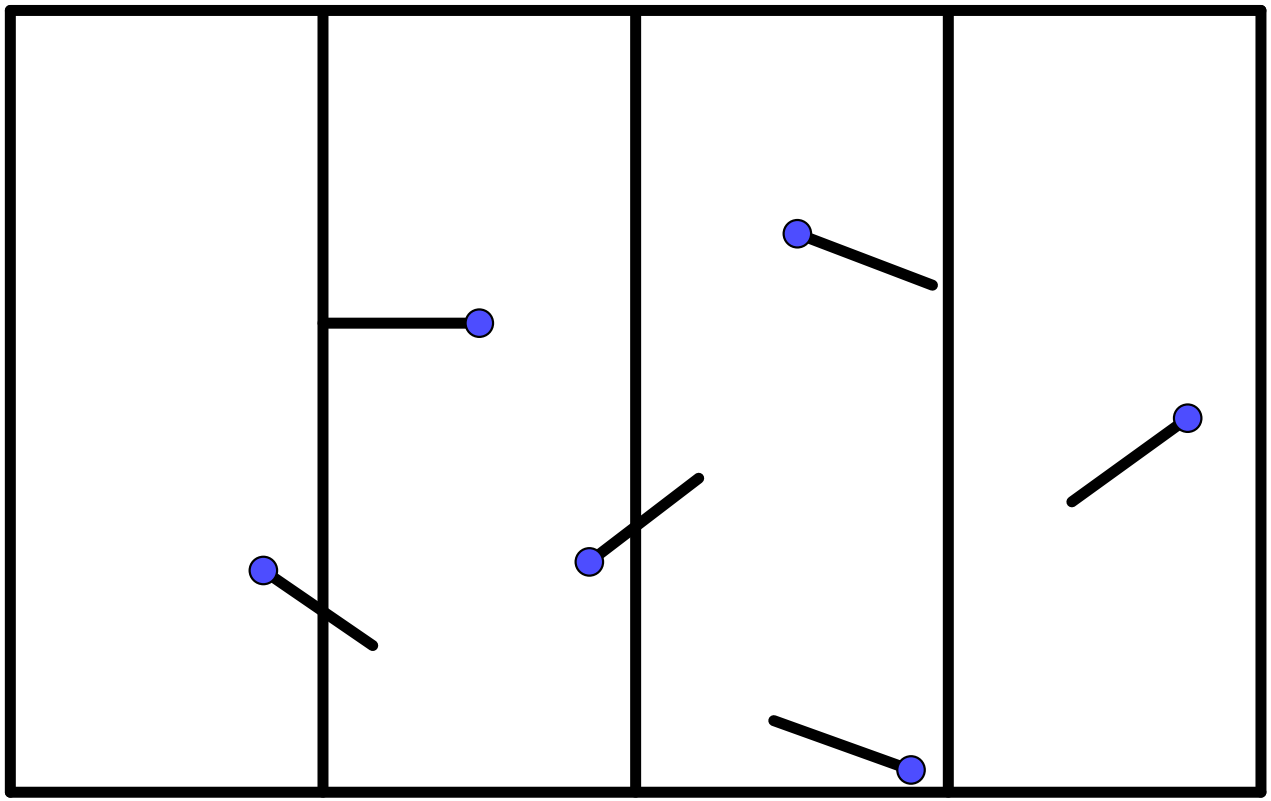
What is the probability of a drop lands in the circle?

$$P(hit) = \frac{\pi r^2}{4r^2} \tag{32}$$



Needles

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



Uniform distribution

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

$$f(x) = \frac{1}{b-a} \tag{33}$$

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

$$F(x) = \int_{-\infty}^{x} f(u)du \tag{34}$$

```
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)

Deck of cards

52 standard

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

Gamblers ruin

Topics to research

- Conditional indepedence
- Panzer ruin
- vander

Matrix

```
matrix = [[1,2,3],[10,11,12],[18,19,29]]
```

Matrix

Sum

```
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')
  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 tranpose

```
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
```

identity

```
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
```

identity

```
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
```

inner product

```
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)
```

Product

```
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
```

Numpy

What is numpy and why it is important? numpy is a open source project, that allow us work with arrays.
np.arrays are most fast than built-in lists.
some functions of numpy are written in C or C++.

Inner(dot) product

$$\vec{u}.\vec{v} = \sum u_i v_i$$
$$\vec{u}.\vec{u} = \frac{n(n+1)(2n+1)}{6} \tag{35}$$

```
a = np.array([1,2,3,4])
b = np.array([1,3,4,4))
np.dot(a,b)

def sum_square(number):
  return number * (number+1) * (2*number +1 ) / 6
sum_square(6)
```

Matrix multiplication

@

When the arrays are of 1-D then uses*np.dot*() otherwise, 2-D arrays uses *np.matmult*() or @.

$$AB = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 5 & 2 \\ 4 & 2 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 4 & 5 \\ 3 & 7 \\ 4 & 2 \end{pmatrix}$$

```
A = np.array([[1,2,3], [1,5,2], [4,2,0]])
B = np.array([[1,0],[0,1],[1,0]])
A @ B
np.matmul(A,B)
```

Numpy

other functions

```
np.zeros((rows,columns)) #Return a array of zeros.
np.transpose(array) # Return the transpose matrix.
np.ones_like(array) # Return the array with the same size but
    filled with ones.
```

Switch

```
def funcion1():
  print('f1')
def funcion2():
  print('f2')
def funcion3():
  print('f3')
def switch(a):
  swithcer={1: funcion1,
            2: funcion2,
            3: funcion3
  }
  return swithcer.get(a)()
switch(1)
```

Duck typing

Dynamic typing

python remember only the last assigment. The last assignment to a 'variable' is the last value.

Type hints

We said previously that is important in object oriented programming, to know, what type of object will be returned by a function, this allow us read more easily the code.
To write better code

```
def factorial(n:int) -> int:
statements..
```

This is a way of write better code, but take time.

Type hints

```
def factorial(n:int) -> int:
acum: int = n
for x in range(1,n):
    acum: int = acum + x
return acum
```

This is a way of write better code, but take time.

To see documentation about type hints.

```
factorial.__annotations__
```

types

`int, float,str, NoneType`

Unicode

is a standard to codify characters, in python the function ord() return its code. ASCII values..

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(7)
```

Split own implementation

```
def split(text, char):
  joint, result = '', []
  for letter in text:
    if letter == char:
      result.append(joint)
      joint = ''
    else:
      joint = joint + letter
  return result
```

Ceaser code criptography

```
def cripto(text,constant):
  result = ''
  for letter in text:
    letter_coded = str(ord(letter)+ constant) + '-'
    result = result + letter_coded
  result = result[0:-1]
  return result

def decifre(text,constant):
  result = ''
  for code in text:
    decode = chr(int(code)-constant)
    result = result + decode
  return result
```

Dynamic vs static typing

we said that the language follow a static programming style if the variables must be defined in compilation time. Otherwise, dynamic programming define variables in execution time.

```
int c = 10
```

strong and weak typing: python have a strong typing for instance

```
a = 10
b = '1'
print(a+b)
```

this will araise a error, in a weak language one variable cast to compatible type for instance to concatenate, '101'.
Python allow us a static system with type hints, for instance;

```
var: float = 10.1
```

mypy

spite of the ability of python to be explicit definition of type, not guaranteed that the parser arise a error if the variable change of type in execution time. we can uses mypy to check the consistency.
if we want a variable that change in the execution then we could define as:

```
from typing import Any
var: Any = 10
```

Then mypy not raises a error.

Delete functions

```
del function_Name
```

Modules and packages

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

```
sys.path # Directories to load modules
```

Namespace

Could exist two or more variables with the same name, living in different namespace. The scope of a variable play a key role: **global** variable could be invoked inside or outside of a function, otherwise a **local** variable only could be invoked inside the function(this imply that the variable was defined in the same function).

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, its scope is local. If two variables have the same name, local variable override the global.

Namespace and scope

variables are identifiers mapping to the objects stored in memory, a dictionary of variables names is called **Namespace**. Each function has its corresponding namespace.

Namespace and scope

In each invocation of a function, it was created a scope, and this is destroyed when appear return.

```
def function(x):
    result = ...
def function_(x):
    result = ...
```

Notice that **result** do not clash or rise a error, due both variables are defined in local scope. this means that is not allowed be invoked from another side of its own scope.

LEGB rule

Local,Enclosing,Global, Built-in

paradigm of Scope this mechanism avoid collision by names, python names could came from: Assignments, import , def and Class.

- Local scope: python functions or lambda expressions
- Enclosing or nolocal: Nested functions
- Global: related to the module, visible in all program.
- Built-in: have keywords, functions, exceptions, that are built in.

This rule is a hierarchical structure for searching or call a variable.

Symbol table

Is a data structure that contain information about; Methods,classes, variables, and so on of a program. there are global and local tables:

```
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; for instance **np.float()** It is important to know that each module have its own symbol table.

dir

the **dir()** function tell us that things are inside of a package.

```
dir(module_name)
```

could be useful to remember methods and attribute inside a package.

__main__

Sometimes you write a module to be imported and its behavior must be change regarding if itself is the main program o will be part of another program.

```
if __name__=='main':
    print('the main program never will be imported')

else:
    print('was loaded and not be a main program')
```

save the last snippet as **two.py**

```
import two
print('this is main and was be executed form shell')
```

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

```
python3
```