

Introducción a la Programación

Anderson Daniel Grajales Alzate

Febrero 25 de 2019

Análisis Numérico / Procesos Numéricos

agrajal7@eafit.edu.co

1. Algoritmos

Definición

Características

2. Pseudocódigo

Definición

Partes

3. Octave

Introducción

Tipos de Datos I

Casting

Documentación

Estructuras de Control

Funciones

Entrada y Salida

4. Python (Opcional)

Introducción

Tipos de Datos I

Casting

Documentación

Estructuras de Control

Funciones

Entrada y Salida

Errores y Excepciones

Tipos de Datos II

Módulos y Paquetes

Técnicas Avanzadas

Computación Numérica

Cálculo Simbólico

Gráficos

Algoritmos

Definición

Secuencia cronológica y ordenada de pasos que llevan a la solución de un problema o a la ejecución de una tarea o actividad.

- **Finito:** Debe terminar después de un número finito de pasos.

- **Finito:** Debe terminar después de un número finito de pasos.
- **Definido:** Cada paso debe estar bien precisado y no debe haber ambigüedad en ninguno de éstos.

- **Finito:** Debe terminar después de un número finito de pasos.
- **Definido:** Cada paso debe estar bien precisado y no debe haber ambigüedad en ninguno de éstos.
- **Entrada:** Debe tener cero o más entradas.

- **Finito:** Debe terminar después de un número finito de pasos.
- **Definido:** Cada paso debe estar bien precisado y no debe haber ambigüedad en ninguno de éstos.
- **Entrada:** Debe tener cero o más entradas.
- **Salida:** Debe tener al menos una salida.

- **Finito:** Debe terminar después de un número finito de pasos.
- **Definido:** Cada paso debe estar bien precisado y no debe haber ambigüedad en ninguno de éstos.
- **Entrada:** Debe tener cero o más entradas.
- **Salida:** Debe tener al menos una salida.
- **Efectivo:** Cada operación debe ser lo suficientemente básica, de tal forma que la ejecución de la misma termine en un tiempo finito.

Pseudocódigo

- Serie de pasos o procedimientos que permiten alcanzar un resultado o resolver un problema.

Pseudocódigo

- Serie de pasos o procedimientos que permiten alcanzar un resultado o resolver un problema.
- Describe un algoritmo utilizando una mezcla de frases en lenguaje común, instrucciones de programación y palabras clave que definen las estructuras básicas.


Pseudocódigo

- Serie de pasos o procedimientos que permiten alcanzar un resultado o resolver un problema.
- Describe un algoritmo utilizando una mezcla de frases en lenguaje común, instrucciones de programación y palabras clave que definen las estructuras básicas.
- No necesariamente siempre es estándar en cada algoritmo que se escribe.

Pseudocódigo

- Serie de pasos o procedimientos que permiten alcanzar un resultado o resolver un problema.
- Describe un algoritmo utilizando una mezcla de frases en lenguaje común, instrucciones de programación y palabras clave que definen las estructuras básicas.
- No necesariamente siempre es estándar en cada algoritmo que se escribe.
- Uso lenguaje natural para expresar el funcionamiento de diferentes procedimientos.

1.2. Método de Steffensen

```
// Tomado de:  
https://en.wikipedia.org/wiki/Steffensen%27s_method  
Leer  $x_0$ ,  $niter$ ,  $tol$   Entrada  
 $error = 1 + tol$   
 $contador = 0$   
mientras  $contador < niter$  y  $error > tol$  hacer  
|  $x_1 = f(x_0)$   
|  $x_2 = f(x_1)$   
|  $f(x) = f(x_0)$   
|  $p = x_0 - \frac{(x_1 - x_2)^2}{x_2 - 2 * x_1 + x_0}$   
|  $error = |x - x_0|$   
|  $x_0 = p$   
|  $contador = contador + 1$   
fin  
si  $error \leq tol$  entonces  
| Hay una raíz en  $p$   
en otro caso  
| El método fracasó después de  $niter$  iteraciones  
fin
```

Algoritmo 2: Método de Steffensen

1.2. Método de Steffensen

```
// Tomado de:  
  https://en.wikipedia.org/wiki/Steffensen%27s_method  
Leer  $x_0$ ,  $niter$ ,  $tol$   
 $error = 1 + tol$   
 $contador = 0$   
mientras  $contador < niter$  y  $error > tol$  hacer  
   $x_1 = f(x_0)$   
   $x_2 = f(x_1)$   
   $f(x) = f(x_0)$   
   $p = x_0 - \frac{(x_1 - x_2)^2}{x_2 - 2 * x_1 + x_0}$   
   $error = |x - x_0|$   
   $x_0 = p$   
   $contador = contador + 1$   
fin  
si  $error \leq tol$  entonces  
  | Hay una raíz en  $p$   
en otro caso  
  | El método fracasó después de  $niter$  iteraciones  
fin
```

Procedimiento

Algoritmo 2: Método de Steffensen

1.2. Método de Steffensen

```
// Tomado de:  
https://en.wikipedia.org/wiki/Steffensen%27s_method  
Leer  $x_0$ ,  $niter$ ,  $tol$   
 $error = 1 + tol$   
 $contador = 0$   
mientras  $contador < niter$  y  $error > tol$  hacer  
|  $x_1 = f(x_0)$   
|  $x_2 = f(x_1)$   
|  $f(x) = f(x_0)$   
|  $p = x_0 - \frac{(x_1 - x_0)^2}{x_2 - 2x_1 + x_0}$   
|  $error = |x - x_0|$   
|  $x_0 = p$   
|  $contador = contador + 1$   
fin  
si  $error \leq tol$  entonces  
| Hay una raíz en  $p$   
en otro caso  
| El método fracasó después de  $niter$  iteraciones  
fin
```

 Salida

Algoritmo 2: Método de Steffensen

Octave

- Lenguaje de programación principalmente diseñado para trabajar con cálculos computacionales.

- Lenguaje de programación principalmente diseñado para trabajar con cálculos computacionales.
- Corre en distintas plataformas.

- Lenguaje de programación principalmente diseñado para trabajar con cálculos computacionales.
- Corre en distintas plataformas.
- Código interpretado.

- Potente en cálculo computacional.

- Potente en cálculo computacional.
- Fácil de usar para este tipo de trabajos.

- Potente en cálculo computacional.
- Fácil de usar para este tipo de trabajos.
- Sintaxis clara.

- Potente en cálculo computacional.
- Fácil de usar para este tipo de trabajos.
- Sintaxis clara.
- Lenguaje común entre los científicos de datos y personas que trabajan con cálculo computacional.

- Potente en cálculo computacional.
- Fácil de usar para este tipo de trabajos.
- Sintaxis clara.
- Lenguaje común entre los científicos de datos y personas que trabajan con cálculo computacional.
- Modularidad completa.

Hello Name!

```
1 clc;  
2 name = "Anderson";  
3 printf("Hello %s\n", name);
```

Hello Name!

```
1 clc;  
2 name = "Anderson";  
3 printf("Hello %s\n", name);
```

Program 1

```
1 clc;  
2 pi = 3;  
3 radius = 11;  
4 area = pi * (radius ** 2);  
5 printf("%.4f\n", area);
```

Program 2

```
1  clc;  
2  name = input("", 's');  
3  printf("Hello %s\n", name);
```

- Lea dos enteros x , y de la entrada estandar e imprima el valor de x^y con una precisión de 5 decimales correctos.

- Existen principalmente tres tipos de datos.

Tipos de Datos I

- Existen principalmente tres tipos de datos.
- Tipos Numéricos: **scalar, vector, matrix, complex.**

Tipos de Datos I

- Existen principalmente tres tipos de datos.
- Tipos Numéricos: **scalar**, **vector**, **matrix**, **complex**.
- Tipos Cadena: **Char**.

Tipos de Datos I

- Existen principalmente tres tipos de datos.
- Tipos Numéricos: **scalar**, **vector**, **matrix**, **complex**.
- Tipos Cadena: **Char**.
- Tipos Estructura de Datos.

Tipos de Datos I

- Existen principalmente tres tipos de datos.
- Tipos Numéricos: **scalar**, **vector**, **matrix**, **complex**.
- Tipos Cadena: **Char**.
- Tipos Estructura de Datos.
- Tipos definidos por el usuario.

Program 3

```
1  clc;
2  integer = int32(3);
3  printf("%d %s\n", integer, class(integer));
4  decdouble = 2;
5  printf("%f %s\n", decdouble, class(decdouble));
6  integer = "3";
7  printf("%s %s\n", integer, class(integer));
8  disp(integer + 11);
```

- Es el acto de pasar de un tipo de objeto a otro.

- Es el acto de pasar de un tipo de objeto a otro.

Program 4

```
1  clc;
2  integer="3";
3  as_integer = str2num(integer);
4  disp(as_integer + 4);
5  boolean = "true";
6  disp(false && str2num(boolean));
7  ddouble = "4.512";
8  disp(str2double(ddouble) + 4.5);
```


- Se puede usar **help()** para obtener ayuda general en línea.

- Se puede usar **help()** para obtener ayuda general en línea.
- Usar **doc()** para una ayuda general más específica.

- Se puede usar **help()** para obtener ayuda general en línea.
- Usar **doc()** para una ayuda general más específica.
- Usar **doc *obj*** para saber información detallada de un tipo de dato u objeto.

- Se puede usar **help()** para obtener ayuda general en línea.
- Usar **doc()** para una ayuda general más específica.
- Usar **doc *obj*** para saber información detallada de un tipo de dato u objeto.

Program 5

```
1 clc;  
2 help();  
3 doc int32;
```

- Los programas se ejecutan de manera **secuencial**.

- Los programas se ejecutan de manera **secuencial**.
- A veces el desarrollador quiere que el programa no siga una secuencia específica sino que tome un *camino* específico.

- Los programas se ejecutan de manera **secuencial**.
- A veces el desarrollador quiere que el programa no siga una secuencia específica sino que tome un *camino* específico.
- Las estructuras de control **if**, **for**, **while**, etc. nos permiten indicarle esos casos al código.

Instrucción if i

if Statement 1

```
1 a = int32(input("Ingrese su edad\n"));
2 if a >= 18
3     disp("Eres mayor de edad");
4 else
5     disp("Eres menor de edad");
6 end
```


Instrucción if ii

if Statement 2

```
1  clc;
2  constant_number = 7;
3  integer = int32(input("Ingrese n\n"));
4  if integer == constant_number
5      disp("Los números son iguales.");
6  elseif integer > constant_number
7      disp("El número ingresado es mayor.");
8  else
9      disp("El número ingresado es menor.");
10 end
```

- Comparación: $=$, $<$, $>$, \leq , \geq , \neq .

- **Comparación:** `==`, `<`, `>`, `<=`, `>=`, `!=`.
- **And/Or:** `a && b`, `a || b`.

Operadores relacionales

- **Comparación:** `==`, `<`, `>`, `<=`, `>=`, `!=`.
- **And/Or:** `a && b`, `a || b`.
- **Not:** `not`

Operadores relacionales

- **Comparación:** ==, <, >, <=, >=, !=.
- **And/Or:** a && b, a || b.
- **Not:** not

Relationals

```
1 clc;  
2 if not(a > b) || (b > c) && not(false)  
3     disp('');  
4 end
```

- Lea dos valores x , y de la entrada estandar e imprima **YES** si existe algún valor a , tal que $y = ax$. En caso contrario imprima **NO**.

Instrucción while

- Esta instrucción ejecuta el contenido que está indentando hasta que la condición evaluada es falsa.

Instrucción while

- Esta instrucción ejecuta el contenido que está indentando hasta que la condición evaluada es falsa.

while Statement

```
1  clc;
2  n = input("");
3  while n > 0
4      printf("%d;", n);
5      n -= 1;
6  end
```


Instrucción for

- Se usa principalmente para iterar sobre **rangos**.

Instrucción for

- Se usa principalmente para iterar sobre **rangos**.
- Es muy útil en los casos donde yo conozco cual es el número de iteraciones fijas.

Instrucción for

- Se usa principalmente para iterar sobre **rangos**.
- Es muy útil en los casos donde yo conozco cual es el número de iteraciones fijas.
- Es la herramienta principal para iterar sobre listas y otras estructuras de datos.

Instrucción for

- Se usa principalmente para iterar sobre **rangos**.
- Es muy útil en los casos donde yo conozco cual es el número de iteraciones fijas.
- Es la herramienta principal para iterar sobre listas y otras estructuras de datos.

for Statement

```
1  clc;  
2  n = int32(input(""));  
3  iters = 1:n;  
4  for i=iters  
5      printf("%d\n", i);  
6  end
```

for Statement 2

```
1 clc;  
2 vec = [2, 3, 4, 5, 6, 7];  
3 for element=vec  
4     printf("%d\n", element);  
5 end
```

for Statement 3

```
1  clc;
2  n = int32(input(""));
3  disp(n);
4  iters = 1:2:n;
5  for i = iters
6      printf("%d\n", i);
7  end
```

- Cuando existen secuencias de código que se repiten varias veces dentro (del código), se pueden agrupar en funciones.

- Cuando existen secuencias de código que se repiten varias veces dentro (del código), se pueden agrupar en funciones.
- Una función recibe cero o más parámetros de entrada y devuelve cero o más resultados.

- Cuando existen secuencias de código que se repiten varias veces dentro (del código), se pueden agrupar en funciones.
- Una función recibe cero o más parámetros de entrada y devuelve cero o más resultados.
- Los cálculos se realizan directamente dentro de la función.

- Cuando existen secuencias de código que se repiten varias veces dentro (del código), se pueden agrupar en funciones.
- Una función recibe cero o más parámetros de entrada y devuelve cero o más resultados.
- Los cálculos se realizan directamente dentro de la función.
- Las variables declaradas dentro de la función no son visibles a otras funciones.

- Cuando existen secuencias de código que se repiten varias veces dentro (del código), se pueden agrupar en funciones.
- Una función recibe cero o más parámetros de entrada y devuelve cero o más resultados.
- Los cálculos se realizan directamente dentro de la función.
- Las variables declaradas dentro de la función no son visibles a otras funciones.
- Son muy útiles para reutilizar código.

- Cuando existen secuencias de código que se repiten varias veces dentro (del código), se pueden agrupar en funciones.
- Una función recibe cero o más parámetros de entrada y devuelve cero o más resultados.
- Los cálculos se realizan directamente dentro de la función.
- Las variables declaradas dentro de la función no son visibles a otras funciones.
- Son muy útiles para reutilizar código.
- ¡Se van a usar ampliamente en el curso!

Functions

```
1 • clc;
2 function [res] = square(n)
3     res = n * n;
4 end
5 function [res] = add(vec_d)
6     sum_d = 0.0;
7     for e=vec_d
8         sum_d += square(e);
9     end
10    res = sum_d;
11 end
12 disp(add([1, 2, 3, 4]));
```

Functions 2

```
1  clc;
2  function [res err] = taylorSin(x, niter);
3      result = 0.0;
4      for i=0:niter;
5          den = factorial(2 * i + 1);
6          num = (-1) ^ i;
7          mult = (x) ^ (2 * i + 1);
8          result = result + (num / den) * mult;
9      endfor
10     res = result;
11 end
12 disp(taylorSin(1/2, 100));
```

Functions 3

```
1  clc;
2  function [res] = matmul(A, B)
3      [m1, n1] = size(A);
4      [m2, n2] = size(B);
5      result = zeros([int32(n1), int32(m2)]);
6      for i=1:m1
7          for j=1:n2
8              for k=1:m1
9                  result(i, j) += A(i, k) * B(k, j);
10             end
11         end
12     end
13     res = result;
14 end
```

- Todos los programas que construimos tiene salida y/o entrada.

- Todos los programas que construimos tiene salida y/o entrada.

Input/Output

```
1 clc;  
2 f = fopen("input.txt", 'r');  
3 data = fread(f);  
4 disp((data(2) - "0"));  
5 fclose(f);
```

Python (Opcional)

- Lenguaje de programación dinámico que soporta diferentes paradigmas de programación.

- Lenguaje de programación dinámico que soporta diferentes paradigmas de programación.
- Código independiente de la plataforma.

- Lenguaje de programación dinámico que soporta diferentes paradigmas de programación.
- Código independiente de la plataforma.
- Código interpretado.

- Lenguaje extremadamente versátil.

- Lenguaje extremadamente versátil.
- Sintaxis clara.

- Lenguaje extremadamente versátil.
- Sintaxis clara.
- Lenguaje común.

- Lenguaje extremadamente versátil.
- Sintaxis clara.
- Lenguaje común.
- Modularidad completa.

- Lenguaje extremadamente versátil.
- Sintaxis clara.
- Lenguaje común.
- Modularidad completa.
- Gran Comunidad.

Hello Name!

```
1 name = "Anderson"
2 if __name__ == "__main__":
3     print("Hello " + name)
```

Primeros Pasos

Hello Name!

```
1 name = "Anderson"
2 if __name__ == "__main__":
3     print("Hello " + name)
```

Program 1

```
1 if __name__ == "__main__":
2     pi = 3
3     radius = 11
4     area = pi * (radius ** 2)
5     print(f"% .4f" % area)
```

Program 2

```
1 name = input()
2 if __name__ == "__main__":
3     print("Hello " + name)
```

- Los **objetos** son el núcleo de las cosas que Python puede manipular.

- Los **objetos** son el núcleo de las cosas que Python puede manipular.
- Cada objeto tiene un tipo.

Tipos de Datos I

- Los **objetos** son el núcleo de las cosas que Python puede manipular.
- Cada objeto tiene un tipo.
- Los tipos son **escalares** o **no-escalares**.

Tipos de Datos I

- Los **objetos** son el núcleo de las cosas que Python puede manipular.
- Cada objeto tiene un tipo.
- Los tipos son **escalares** o **no-escalares**.
- Los tipos escalares son: **int**, **float**, **bool** y **None**.

Tipos de Datos I

- Los **objetos** son el núcleo de las cosas que Python puede manipular.
- Cada objeto tiene un tipo.
- Los tipos son **escalares** o **no-escalares**.
- Los tipos escalares son: **int**, **float**, **bool** y **None**.
- Los objetos y los **operadores** pueden usarse para formar **expresiones**.

Program 3

```
1 if __name__ == "__main__":  
2     integer = 3  
3     print(integer, type(integer))  
4     double = 3.4  
5     print(double, type(double))  
6     integer = "3"  
7     print(integer, type(integer))  
8     print(integer + 11)
```

Program 4

```
1 if __name__ == "__main__":
2     integer = "3"
3     as_integer = int(integer)
4     print(as_integer + 4)
5     boolean = "True"
6     print(False and bool(boolean))
7     double = "4.512"
8     print(float(double) + 4.5)
```

- Se puede usar **help()** para obtener ayuda general en línea.

- Se puede usar **help()** para obtener ayuda general en línea.
- Se puede usar **help(*obj*)** para consultar sobre un objeto específico.

Documentación

- Se puede usar **help()** para obtener ayuda general en línea.
- Se puede usar **help(*obj*)** para consultar sobre un objeto específico.
- Usar **dir()** para saber todos los nombres usados.

- Se puede usar **help()** para obtener ayuda general en línea.
- Se puede usar **help(obj)** para consultar sobre un objeto específico.
- Usar **dir()** para saber todos los nombres usados.
- Usar **dir(obj)** para saber todos los nombres usados por un objeto.

Documentación

- Se puede usar **help()** para obtener ayuda general en línea.
- Se puede usar **help(obj)** para consultar sobre un objeto específico.
- Usar **dir()** para saber todos los nombres usados.
- Usar **dir(obj)** para saber todos los nombres usados por un objeto.

Program 5

```
1 if __name__ == "__main__":  
2     # print(help())  
3     print(help(int))  
4     print(dir())  
5     print(dir(int))
```

Instrucción if i

if Statement 1

```
1 if __name__ == "__main__":  
2     a = int(input("Ingrese su edad\n"))  
3     if a >= 18:  
4         print("Eres mayor de edad.")  
5     else:  
6         print("Eres menor de edad.")
```

Instrucción if ii

if Statement 2

```
1 if __name__ == "__main__":
2     constant_number = 7
3     integer = int(input("Ingrese n:\n"))
4     if integer == constant_number:
5         print("Los números son iguales.")
6     elif integer > constant_number:
7         print("El número ingresado es mayor.")
8     else:
9         print("El número ingresado es menor.")
```

- **Comparación:** $==$, $<$, $>$, $<=$, $>=$, $!=$.

Operadores relacionales

- Comparación: $==$, $<$, $>$, $<=$, $>=$, $!=$.
- Comparación de objetos: a is b , a is not b .

Operadores relacionales

- **Comparación:** $==$, $<$, $>$, $<=$, $>=$, $!=$.
- **Comparación de objetos:** a is b , a is not b .
- **And/Or:** a and b , a or b .

Operadores relacionales

- **Comparación:** $==$, $<$, $>$, $<=$, $>=$, $!=$.
- **Comparación de objetos:** a is b , a is not b .
- **And/Or:** a and b , a or b .
- **Not:** not

Operadores relacionales

- Comparación: `==`, `<`, `>`, `<=`, `>=`, `!=`.
- Comparación de objetos: `a is b`, `a is not b`.
- And/Or: `a and b`, `a or b`.
- Not: `not`

Relationals

```
1 if not(a > b) or (b > c) and not(False):  
2     pass
```


Instrucción while

while Statement

```
1 if __name__ == "__main__":  
2     n = int(input())  
3     while n > 0:  
4         print(n, end=';')  
5         n -= 1
```

for Statement

```
1 if __name__ == "__main__":  
2     n = int(input())  
3     iters = range(1, n)  
4     for i in iters:  
5         print(i)
```

for Statement 2

```
1 if __name__ == "__main__":  
2     linklist = [2, 3, 4, 5, 6, 7]  
3     for i in linklist:  
4         print(i)
```

for Statement 3

```
1 if __name__ == "__main__":  
2     n = int(input())  
3     iters = range(1, n, 2)  
4     for i in iters:  
5         print(i)
```

Functions

```
1 ● def square(n):  
2     return n * n  
3 def add(linklist):  
4     sum = 0  
5     for s in linklist:  
6         sum += square(s)  
7     return sum  
8 if __name__ == "__main__":  
9     print(add([1, 2, 3, 4]))
```

Functions 2

```
1 import math
2 def taylorSin(at, maxtoler, maxiter):
3     result , real_value = 0.0, math.sin(at)
4     for i in range(0, maxiter):
5         den = math.factorial(2 * i + 1)
6         num, mult = (-1) ** i, at ** (2 * i + 1)
7         result += (num / den) * mult
8         if abs(real_value - result) < maxtoler:
9             break
10    return result
11 if __name__ == "__main__":
12    print(taylorSin(7/8, 10**-8, 100))
```

Functions 3

```
1 import numpy as np
2 def matmult(A = [], B = []):
3     result = np.zeros((len(A[0]), len(B)))
4     for i in range(len(A)):
5         for j in range(len(B[0])):
6             for k in range(len(A)):
7                 result[i][j] += A[i][k] * B[k][j]
8     return (A, B, result)
9 if __name__ == "__main__":
10     print(matmult([[1, 2], [2, 3]], [[3, 5], [4, 6]]))
```

- Cuando nuestro código se está ejecutando, pueden surgir errores durante la ejecución.

- Cuando nuestro código se está ejecutando, pueden surgir errores durante la ejecución.
- División por cero.

- Cuando nuestro código se está ejecutando, pueden surgir errores durante la ejecución.
- División por cero.
- Leer un archivo que no existe.

- Cuando nuestro código se está ejecutando, pueden surgir errores durante la ejecución.
- División por cero.
- Leer un archivo que no existe.
- Escribir en un archivo de solo lectura.

- Cuando nuestro código se está ejecutando, pueden surgir errores durante la ejecución.
- División por cero.
- Leer un archivo que no existe.
- Escribir en un archivo de solo lectura.
- ... ,

Errores y Excepciones Continuación

Input/Output

```
1  if __name__ == "__main__":
2      a, b = [2, 0]
3      try:
4          print(a/b)
5      except ZeroDivisionError as e:
6          print(e)
7      finally:
8          print("Always execute")
9      try:
10         f = open("doesnotexists.txt", "r")
11     except Exception as e:
12         print(e)
```

- []. Lista de elementos.

- []. Lista de elementos.
- **List**: Estructura ordenada, admite elementos repetidos.

- `[]`. Lista de elementos.
- **List**: Estructura ordenada, admite elementos repetidos.
- **Set**: Estructura no ordenada, no hay elementos duplicados.

- `[]`. Lista de elementos.
- **List**: Estructura ordenada, admite elementos repetidos.
- **Set**: Estructura no ordenada, no hay elementos duplicados.
- **Dictionary**: Estructura no ordenada, la forma de acceder y escribir es *clave, valor*. $f(x) = y$.

Structures

```
1 if __name__ == "__main__":
2     s = set([2, 3, 4, 1, 2, 3, 7])
3     d = dict({"a" : 1, "b" : 2, 2 : 3, 5 : 7})
4     l = list([2, 3, 4, 1, 2])
5     print(set([1, 2, 3]).issubset(s))
6     print((d[2], d["a"]))
7     print(set([8, 1, 2, 3]).difference(s))
8     print(l.count(2))
```

Modules

```
1 import numpy as np # Computación Numérica
2 import sympy as sp # Computación Simbólica
3 import math        # Operaciones matemáticas.
4 import matplotlib  # Graficos
5 import sys         # Sistema
6 if __name__ == "__main__":
7     x = sp.symbols('x')
8     fx = x ** 2 + sp.sin(x)
9     print(fx.diff(x))
10    print(np.random.normal((4, 4)))
```

- Sliding

- Sliding
- Condicionales

- Sliding
- Condicionales
- **Agrupación**

- Sliding
- Condicionales
- Agrupación
- Listas

- Sliding
- Condicionales
- Agrupación
- Listas
- **Simplificación**

Advanced

```
1 import numpy as np
2 if __name__ == "__main__":
3     (a, b) = input().split(" ")
4     result = a if a < b else b
5     print(result)
6     arr = np.arange(100)
7     print(arr[2:len(arr):2])
8     net = [i for i in range(1, 100) if i % 2 != 0]
9     print(net)
```

- Cálculos lineales y no lineales.

- Cálculos lineales y no lineales.
- Operaciones matriciales.

- Cálculos lineales y no lineales.
- Operaciones matriciales.
- Precisión Numérica.

- Cálculos lineales y no lineales.
- Operaciones matriciales.
- Precisión Numérica.
- Simplificación dirigida.

- Cálculos lineales y no lineales.
- Operaciones matriciales.
- Precisión Numérica.
- Simplificación dirigida.
- Optimización.

- Cálculos lineales y no lineales.
- Operaciones matriciales.
- Precisión Numérica.
- Simplificación dirigida.
- Optimización.
- **numpy**

- Cálculos lineales y no lineales.
- Operaciones matriciales.
- Precisión Numérica.
- Simplificación dirigida.
- Optimización.
- **numpy**
- ...

Numeric

```
1 import numpy as np
2 if __name__ == "__main__":
3     a = [2, 3, 5.6, 1, 7.8, 9, 100]
4     np_a = np.array(a)
5     print((np.sum(np_a), np.max(np_a), np.min(np_a)))
6     print(np_a ** 2)
7     print(np.dot(np_a, np_a ** (1/2)))
8     print(np_a[1:5])
```

- Diferenciación.

- Diferenciación.
- Integración.

- Diferenciación.
- Integración.
- Simplificación.

- Diferenciación.
- Integración.
- Simplificación.
- Correlación.

- Diferenciación.
- Integración.
- Simplificación.
- Correlación.
- Evaluación.

- Diferenciación.
- Integración.
- Simplificación.
- Correlación.
- Evaluación.
- **sympy**

- Diferenciación.
- Integración.
- Simplificación.
- Correlación.
- Evaluación.
- **sympy**
- ...

Symbolic

```
1 import sympy as sp
2 if __name__ == "__main__":
3     x = sp.symbols('x')
4     poly = 3 * x**4 + 4 * x**3 + 2 * x**2 - 4
5     dpoly = poly.diff(x)
6     print(dpoly)
7     print(poly.evalf(subs={x: 7}))
8     ipoly = poly.integrate(x)
9     print(ipoly)
10    print(ipoly.evalf(subs={x: 7}))
```

- 1D

- 1D
- 2D

- 1D
- 2D
- 3D

- 1D
- 2D
- 3D
- Intersección.

- 1D
- 2D
- 3D
- Intersección.
- Aproximación.

- 1D
- 2D
- 3D
- Intersección.
- Aproximación.
- **matplotlib.**

- 1D
- 2D
- 3D
- Intersección.
- Aproximación.
- **matplotlib.**
- ...

Graphs

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 if __name__ == "__main__":
4     x = np.linspace(-2, 2, num=100)
5     y = x ** 2
6     plt.plot(x, y)
7     plt.xlabel("x")
8     plt.ylabel("y")
9     plt.show()
```

Gracias!