Ing. Ventre, Luis O.

CAPITULO 4:

ESTRUCTURAS DE CONTROL CONT.

- En el ultimo programa visto de **MENU DE USUARIO**, si el usuario ingresaba una opción INVALIDA que pasaba?
- Se le avisaba y luego?...
- Correcto sería, darle la opción de reingresar una **OPCION VALIDA**
- Podría implementarse con una estructura IF?…. «si ingreso una opción errónea solicitar el reingreso»…funcionara?
- Necesitamos una instrucción de REPETICION de sección de código.

Sentencia WHILE

- Esta instrucción implica:
  - Mientras se cumpla esta condición, repite estas acciones.
- Y su sintaxis:

while *condición*:
acción
acción
acción
….
acción

- Las sentencias que denotan repetición se denominan **BUCLES**.

- Analicemos la traza del siguiente programa:

contador_3.py

```
1 i = 0
2 while i < 3:
3     print i
4     i += 1
5 print 'Hecho'
```

contador.py

```
0
1
2
Hecho
```

- Puede determinar la salida del programa?....

- **Veamos el porque?.**

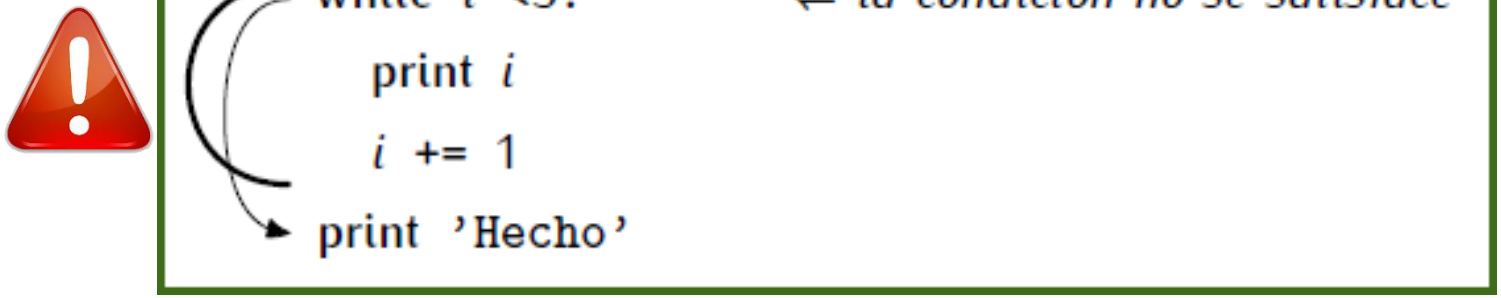
contador_3.py — contador.py

```
1 i = 0
2 while i < 3:
3     print i
4     i += 1
5 print 'Hecho'
```

- **La línea 2, pregunta mientras la condición sea cierta ejecutar lo que esta indentado.**

```
i = 0
while i <3:          ⇐ la condición se satisface
    print i
    i += 1
print 'Hecho'
```

- Las instrucciones internas del BUCLE WHILE se continúan EJECUTANDO hasta que la «condición» sea FALSA.



```
i = 0
while i <3:          ⇐ la condición no se satisface
    print i
    i += 1
print 'Hecho'
```

- En ese momento se saltean las instrucciones internas del BUCLE y se ejecuta el ultimo «print».

Sentencia WHILE

- Pero para que complicarnos, este programa no hace lo mismo?:

contador_simple.py

```
1 print 0
2 print 1
3 print 2
4 print 'Hecho'
```

- Entonces ahora haga lo mismo con el siguiente bucle:

contador_4.py

```
1 i = 0
2 while i < 1000:
3     print i
4     i += 1
5 print 'Hecho'
```

contador.py

Trazas

- Analiza las trazas y salidas de los siguientes ejercicios:

ejercicio_bucle_9.py

```
1 i = 0
2 while i <= 3:
3     print i
4     i += 1
5 print 'Hecho'
```

ejercicio_bucle.py

```
0
1
2
3
Hecho
```

ejercicio_bucle_10.py

```
1 i = 0
2 while i < 10:
3     print i
4     i += 2
5 print 'Hecho'
```

ejercicio_bucle.py

```
0
2
4
6
8
Hecho
```

ejercicio_bucle_13.py

```
1 i = 10
2 while i < 2:
3     i *= 2
4     print i
```

ejercicio_bucle.py

BUCLES SIN FIN

- Analiza el siguiente programa, y estudia detenidamente su TRAZA

bucle_infinito.py

```
1 i = 0
2 while i < 10:
3     print i
```

- Los bucles son muy UTILES, pero debes TENER CUIDADO.

- En el interior del BUCLE
  - **Se debe FALSIFICAR la «condición» para salir del mismo.**

- Piense como resolvería el problema, si le solicita que realice un programa que SUME los primeros 1000 números!...

- Algunas posibles soluciones….elegantes?

sumatorio.py

```
1 sumatorio = 1 + 2 + 3 + ... + 999 + 1000
2 print sumatorio
```

```
1 sumatorio = 0
2 sumatorio += 1
3 sumatorio += 2
4 sumatorio += 3
  :
1000 sumatorio += 999
1001 sumatorio += 1000
1002 print sumatorio
```

Sumatorios

- En el ultimo programa, las instrucciones 1 hasta la 1001 son de la forma:

```
sumatorio += numero
```

- Podríamos usar un bucle, para REPETIR ESTA INSTRUCCION?

sumatorio.py

```
1 sumatorio = 0
2 while condición:
3     sumatorio += número
4 print sumatorio
```

Sumatorios

- Pero numero ha de tomar valores del 1 al 1000. Podríamos utilizar una variable…

sumatorio.py

```
1 sumatorio = 0
2 i = 1
3 while condición:
4     sumatorio += i
5     i += 1
6 print sumatorio
```

- Solo resta indicar la CONDICION que decide si seguir iterando o salir del bucle?....

Sumatorios

- Podríamos resolver en 6 líneas de código, las mil anteriores!.

sumatorio_4.py — sumatorio.py

```
1 sumatorio = 0
2 i = 1
3 while i <= 1000:
4     sumatorio += i
5     i += 1
6 print sumatorio
```

- El siguiente programa hace lo mismo?

sumatorio_5.py — sumatorio.py

```
1 sumatorio = 0
2 i = 0
3 while i < 1000:
4     i += 1
5     sumatorio += i
6 print sumatorio
```

Requisitos en la entrada

- Veamos otro ejemplo útil….

- Analice el siguiente programa, que hace?

raiz_4.py

raiz.py

```
1 from math import sqrt
2
3 x = float(raw_input('Introduce un número positivo: '))
4
5 print 'La raíz cuadrada de %f es %f' % (x, sqrt(x))
```

- Que pasa si el usuario ingresa un numero negativo?....

Requisitos en la entrada

- Como resolveríamos esto, para continuar solicitando un numero cada vez que INGRESA UN NUMERO NEGATIVO?.

raiz.py

```
1 from math import sqrt
2
3 while condición:
4     x = float(raw_input('Introduce un número positivo: '))
5
6 print 'La raíz cuadrada de %f es %f' % (x, sqrt(x))
```

- Y la condición?....

Requisitos en la entrada

- Podría ser, mientas el valor ingresado SEA NEGATIVO…

raiz_5.py — raiz.py

```
1 from math import sqrt
2
3 while x < 0:
4     x = float(raw_input('Introduce un número positivo: '))
5
6 print 'La raíz cuadrada de %f es %f' % (x, sqrt(x))
```

- Esto genera el siguiente error, comprende por que?...

```
Traceback (innermost last):
  File 'raiz.py', line 3, in ?
    while x < 0:
NameError: x
```

Requisitos en la entrada

- La variable X, no esta definida ni inicializada al momento de consultar la «condición». Como podemos solucionarlo?

raiz.py

```
1 from math import sqrt
2
3 x = -1
4 while x < 0:
5     x = float(raw_input('Introduce un número positivo: '))
6
7 print 'La raíz cuadrada de %f es %f' % (x, sqrt(x))
```

- Alguna opción mas elegante?...

- Analiza la siguiente solución:

raiz_6.py

raiz.py

```
1 from math import sqrt
2
3 x = float(raw_input('Introduce␣un␣número␣positivo:␣'))
4 while x < 0:
5     x = float(raw_input('Introduce␣un␣número␣positivo:␣'))
6
7 print 'La␣raíz␣cuadrada␣de␣%f␣es␣%f' % (x, sqrt(x))
```

- F
C
I
C

de

circulo_14.py

circulo.py

```
1  from math import pi
2
3  radio = float(raw_input('Dame el radio de un círculo: '))
4
5  opcion = ''
6  while opcion < 'a' or opcion > 'c':
7      print 'Escoge una opción: '
8      print 'a) Calcular el diámetro.'
9      print 'b) Calcular el perímetro.'
10     print 'c) Calcular el área.'
11     opcion = raw_input('Teclea a, b o c y pulsa el retorno de carro: ')
12     if opcion == 'a':
13         diametro = 2 * radio
14         print 'El diámetro es', diametro
15     elif opcion == 'b':
16         perimetro = 2 * pi * radio
17         print 'El perímetro es', perimetro
18     elif opcion == 'c':
19         area = pi * radio ** 2
20         print 'El área es', area
21     else:
22         print 'Sólo hay tres opciones: a, b o c. Tú has tecleado', opcion
```

- Analiza, que ahora además incluye una alternativa de ingreso de letra para SALIR DEL PROGRAMA.

```
6  while opcion != 'd':
7      print 'Escoge una opción: '
8      print 'a) Calcular el diámetro.'
9      print 'b) Calcular el perímetro.'
10     print 'c) Calcular el área.'
11     print 'd) Finalizar.'
12     opcion = raw_input('Teclea a, b o c y pulsa el retorno de carro: ')
13     if opcion == 'a':
14         diametro = 2 * radio
15         print 'El diámetro es', diametro
16     elif opcion == 'b':
17         perimetro = 2 * pi * radio
18         print 'El perímetro es', perimetro
19     elif opcion == 'c':
20         area = pi * radio ** 2
21         print 'El área es', area
22     elif opcion != 'd':
23         print 'Sólo hay cuatro opciones: a, b, c o d. Tú has tecleado', opcion
24
25 print 'Gracias por usar el programa'
```

El bucle FOR-IN

- Otro instrucción de BUCLE que ofrece python es **for-in**.
- La misma implica «para todo elemento de una serie hacer»
- Su sintaxis:

```
for variable in serie de valores:
    acción
    acción
    acción
    ....
    acción
```

- Para «variable» que toma cada uno de los valores de la «serie» ejecutar todas las instrucciones indentadas.

El bucle FOR-IN

- Veamos ejemplos:

saludos.py

```
1 for nombre in ['Pepe', 'Ana', 'Juan']:
2     print 'Hola, %s.' % nombre
```

```
Hola, Pepe.
Hola, Ana.
Hola, Juan.
```

potencias.py

```
1 numero = int(raw_input('Dame un número: '))
2
3 for potencia in [2, 3, 4, 5]:
4     print '%d elevado a %d es %d' % (numero, potencia, numero ** potencia)
```

El bucle FOR-IN

- Veamos ejemplos:

potencias.py

```
potencias.py
numero = int(raw_input('Dame un número: '))

for potencia in [2, 3, 4, 5]:
    print '%d elevado a %d es %d' % (numero, potencia, numero ** p
```

El bucle FOR-IN

- Cual seria la mejor solución, si quiero que la variable del bucle FOR, tome valores entre 1 y 100?

- Para esto existe una función PREDEFINIDA en python que se llama **RANGE**.

- Esta en su forma mas simple, se utiliza con 2 argumentos, un valor inicial y un valor final.

- Ejemplos:

```
>>> range(2, 10) ↲
[2, 3, 4, 5, 6, 7, 8, 9]
>>> range(0, 3) ↲
[0, 1, 2]
```

El bucle FOR-IN

- Observa que los valores devueltos:
  - **INCLUYEN** al PRIMER elemento de la lista.
  - **NO INCLUYEN** el ULTIMO elemento de la lista.

- Cual será la salida del siguiente programa?...

contador_con_for.py

```
1 for i in range(1, 6):
2     print i
```

contador_con_for.py

- OBI WAN (Kenobi) ---- OFF BY ONE.

La función RANGE

- Podemos usar la función RANGE, con 1, 2 o 3 argumentos.

- La función RANGE con 1 argumento,**por omisión INICIA en 0**.

```
>>> range(5) ↲
[0, 1, 2, 3, 4]
```

- La función RANGE con 3 argumentos, indica, comienzo, fin, y el incremento (salto) para la serie.

```
>>> range(2, 10, 2) ↲
[2, 4, 6, 8]
```

```
>>> range(10, 5, -1) ↲
[10, 9, 8, 7, 6]
```

Bucle WHILE o FOR-IN

- Hay casos donde es mas legible utilizar el bucle FOR-IN, veamos el siguiente caso, de un programa que debe sumar los primeros 1000 números.

sumatorio_6.py — sumatorio.py

```
1 sumatorio = 0
2 i = 1
3 while i <= 1000:
4     sumatorio += i
5     i += 1
6 print sumatorio
```

- En estos casos, la variable «i» suele denominarse «índice» del bucle.

Bucle WHILE o FOR-IN

- Como quedaría este mismo programa con un bucle for-in

sumatorio.py

```
1 sumatorio = 0
2 for i in range(1, 1001):
3     sumatorio += i
4
5 print sumatorio
```

- Comparando:

```
sumatorio = 0
i = 1
while i <= 1000:
    sumatorio += i
    i += 1
print sumatorio
```

```
sumatorio = 0

for i in range (1,1001):
    sumatorio += i

print sumatorio
```

Números PRIMOS

- El objetivo ahora es construir un programa que determine si un numero entero es o no primo.

- Una forma de saberlo, es dividir el numero por todos los valores menores a el, y corroborar que el resto nunca es 0. Lo que equivale a decir que el numero es solo divisible por 1 y por si mismo!.

Dividendo	Divisor	Cociente	Resto
7	2	3	1
7	3	2	1
7	4	1	3
7	5	1	2
7	6	1	1

Números PRIMOS

- Una forma de codificar esto, podría ser, sumar la cantidad de resultados diferentes a 0, y compararla con (numero – 2).

es_primo_11.py

es_primo.py

```
1  num = 7
2
3  restos_no_nulos = 0
4  for divisor in range(2, num):
5      if num % divisor != 0:
6          restos_no_nulos += 1
7
8  if restos_no_nulos == num - 2:
9      print 'El número', num, 'es primo'
10 else:
11     print 'El número', num, 'no es primo'
```

Números PRIMOS

- Una idea mas ELEGANTE y muy USADA consiste en:
  - Asumir una condición como verdadera antes de comenzar el programa. Darle valor «True» a una variable.
  - Recorre todos los posibles casos, si alguno no se cumple, modifica la variable anterior a «False»
  - Al final, chequear la condición, si es True, nadie la modifico, sino alguien la modifico y ya no es primo…

Números PRIMOS

- Veamos como queda el ejercicio:

es_primo_12.py

es_primo.py

```
1  num = 7
2
3  creo_que_es_primo = True
4  for divisor in range(2, num):
5      if num % divisor == 0:
6          creo_que_es_primo = False
7
8  if creo_que_es_primo:
9      print 'El número', num, 'es primo'
10 else:
11     print 'El número', num, 'no es primo'
```

Números PRIMOS

- El siguiente ejercicio esta MAL, entiendo porque?

```
1  num = 7
2
3  creo_que_es_primo = True
4  for divisor in range(2, num):
5      if num % divisor == 0:
6          creo_que_es_primo = False
7      else:
8          creo_que_es_primo = True
9
10 if creo_que_es_primo:
11     print 'El número', num, 'es primo'
12 else:
13     print 'El número', num, 'no es primo'
```

Números PRIMOS

- Si queremos modificar el ejemplo para que el NUMERO a verificar lo ingrese el usuario?

es_primo_14.py

es_primo.py

```
1  num = int(raw_input('Dame un número: '))
2
3  creo_que_es_primo = True
4  for divisor in range(2, num):
5      if num % divisor == 0:
6          creo_que_es_primo = False
7
8  if creo_que_es_primo:
9      print 'El número', num, 'es primo'
10 else:
11     print 'El número', num, 'no es primo'
```

es_primo_14.py

es_primo.py

```
1  num = int(raw_input('Dame un número: '))
2
3  creo_que_es_primo = True
4  for divisor in range(2, num):
5      if num % divisor == 0:
6          creo_que_es_primo = False
7
8  if creo_que_es_primo:
9      print 'El número', num, 'es primo'
10 else:
11     print 'El número', num, 'no es primo'
```

Números PRIMOS

- Podemos optimizar con el siguiente BUCLE WHILE!

es_primo_16.py

es_primo.py

```
1  num = int(raw_input('Dame un número: '))
2
3  creo_que_es_primo = True
4  divisor = 2
5  while divisor < num and creo_que_es_primo:
6      if num % divisor == 0:
7          creo_que_es_primo = False
8      divisor += 1
9
10 if creo_que_es_primo:
11     print 'El número', num, 'es primo'
12 else:
13     print 'El número', num, 'no es primo'
```

Para TODOS / Para Alguno

- Veras que es muy útil, armar bucles para comprobar condiciones para todos.

```
1 creo_que_se_cumple_para_todos = True
2 for elemento in conjunto:
3     if not condición:
4         creo_que_se_cumple_para_todos = False
5
6 if creo_que_se_cumple_para_todos:
7     print 'Se cumple para todos'
```

Para TODOS / Para Alguno

- Veras que es muy útil, armar bucles para comprobar condiciones para alguno.

```
1 creo_que_se_cumple_para_alguno = False
2 for elemento in conjunto:
3     if condición:
4         creo_que_se_cumple_para_alguno = True
5
6 if creo_que_se_cumple_para_alguno:
7     print 'Se cumple para alguno'
```

Rotura de BUCLES - BREAK

- El lenguaje, ofrece una instrucción para FINALIZAR inmediatamente un BUCLE.
- La misma es la instrucción BREAK.
- Puede usarse en bucles WHILE tanto como en FOR – IN
- No debes abusar del BREAK, ya que el mismo supone una forma de salida de bucle FUERA de la forma NATURAL que implica la condicion del while y el valor final del for-in.
- Veamos unos ejemplos.

- Veamos el caso del ultimo programa:

es_primo_17.py

es_primo.py

```
1  num = int(raw_input('Dame un número: '))
2
3  creo_que_es_primo = True
4  divisor = 2
5  while divisor < num:
6      if num % divisor == 0:
7          creo_que_es_primo = False
8          break
9      divisor += 1
10
11 if creo_que_es_primo:
12     print 'El número', num, 'es primo'
13 else:
14     print 'El número', num, 'no es primo'
```

- Y en el caso del for-in:

es_primo_18.py

es_primo.py

```
1  num = int(raw_input('Dame un número: '))
2
3  creo_que_es_primo = True
4  for divisor in range(2, num):
5      if num % divisor == 0:
6          creo_que_es_primo = False
7          break
8
9  if creo_que_es_primo:
10     print 'El número', num, 'es primo'
11 else:
12     print 'El número', num, 'no es primo'
```

Anidamiento de ESTRUCTURAS

- Como es lógico, podemos colocar un for-in, adentro de una estructura for-in….puedes seguir la TRAZA?.

```
for var1 in range(2):
    print 'var1 es',var1
    for var2 in range(3):
        print 'var2 es ahora',var2
```

- La salida será?....

```
var1 es 0
var2 es ahora 0
var2 es ahora 1
var2 es ahora 2
var1 es 1
var2 es ahora 0
var2 es ahora 1
var2 es ahora 2
```

Anidamiento de ESTRUCTURAS

- Como modificaría el programa de los números primos para solicitar al usuario un valor limite para chequear todos los NUMEROS PRIMOS hasta ese valor?

primos.py

```
1  limite = int(raw_input('Dame un número: '))
2
3  for num in range(1, limite+1):
4      creo_que_es_primo = True
5      for divisor in range(2, num):
6          if num % divisor == 0:
7              creo_que_es_primo = False
8              break
9      if creo_que_es_primo:
10         print num
```

Break, solo rompe 1 ciclo

Anidamiento de ESTRUCTURAS

- Veamos un ejemplo mas:

- Implemente un bucle for in, que imprima los días de la semana, considerando 1 a Lunes, 2 a Martes…

- Luego implemente otro bucle for in, que imprima los números de las semanas del mes, semana1, semana2…

- Analice con cuidado, que bucle debe ser externo y que bucle debe ser interno RECORDANDO:

  - El bucle interno deberá ejecutarse completamente antes que el bucle externo avance un paso.

Anidamiento de ESTRUCTURAS

- La solución:

```
>>> for j in range (1,5):
...     print 'semana',j
...     for i in range (1,8):
...         print 'dia',i
...
```

```
semana 1
dia 1
dia 2
dia 3
dia 4
dia 5
dia 6
dia 7
semana 2
dia 1
dia 2
dia 3
dia 4
dia 5
```

- En los programas que hemos realizado, puedes observar que se pueden producir errores en tiempo de EJECUCION.

- Estos son EXCEPCIONES.

- Ej.

Divisiones por 0

Raíces de números negativos

Operar distintos tipos de datos.

- Hasta el momento, lo hemos solucionado con IFs.

- Pero hay una estructura de control especial para la DETECCION y TRATAMIENTO de EXCEPCIONES.

try-except

- Y su uso:

```
try:
    acción potencialmente erronea
    acción potencialmente erronea
    acción potencialmente erronea
except:
    acción para tratar el error
    acción para tratar el error
```

Captura y tratamiento de EXCEPCIONES

- Podría traducirse a:

  «Intenta ejecutar estas instrucciones si se produce un error, ejecuta inmediatamente estas otras».

- Veamos como aplica al ejercicio de resolución de ecuación de primer grado:

primer_grado_14.py — primer_grado.py

```
1  a = float(raw_input('Valor de a: '))
2  b = float(raw_input('Valor de b: '))
3
4  try:
5      x = -b/a
6      print 'Solución: ', x
7  except:
8      if b != 0:
9          print 'La ecuación no tiene solución.'
10     else:
11         print 'La ecuación tiene infinitas soluciones.'
```

Captura y tratamiento de EXCEPCIONES

- Y si hay mas de un posible error? en el **mismo bloque try**?

- Se pueden poner mas de un bloque **EXCEPT**.

- Cada uno tratara un tipo de error diferente!....veamos los tipos de errores:

```
>>> 1 / 0 ↲
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
>>> from math import sqrt ↲
>>> sqrt(-1) ↲
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ValueError: math domain error
```

- El programa de solución ec. de segundo grado quedaría:

segundo_grado_24.py

segundo_grado.py

```
1  from math import sqrt
2
3  a = float(raw_input('Valor de a: '))
4  b = float(raw_input('Valor de b: '))
5  c = float(raw_input('Valor de c: '))
6
7  try:
8      x1 = (-b + sqrt(b**2 - 4*a*c)) / (2 * a)
9      x2 = (-b - sqrt(b**2 - 4*a*c)) / (2 * a)
10     if x1 == x2:
11         print 'Solución de la ecuación: x=%4.3f' % x1
12     else:
13         print 'Soluciones de la ecuación: x1=%4.3f y x2=%4.3f' % (x1, x2)
14 except ZeroDivisionError:
15     if b != 0:
16         print 'La ecuación no tiene solución.'
17     else:
18         print 'La ecuación tiene infinitas soluciones.'
19 except ValueError:
20     print 'No hay soluciones reales'
```

- Todas las instrucciones vistas, pueden utilizarse en conjunto con el lienzo para dibujar, esto permitirá encontrar importantes aplicaciones.

- Por ejemplo, imagine un programa que grafique la función seno?.

- Pueden ser necesarias las funciones:
  - window_coordinates(x1,y1,x2,y2). Cambia el sistema de coordenadas.

  - window_size(x,y). Cambia el tamaño del lienzo. Pasa a ser de x por y pixels.

- Veamos una primera aproximación del programa para graficar funciones!.

seno_6.py

seno.py

```
1 from math import pi, sin
2
3 window_size (500, 500)
4 window_coordinates(-2*pi, -1.5, 2*pi, 1.5)
```

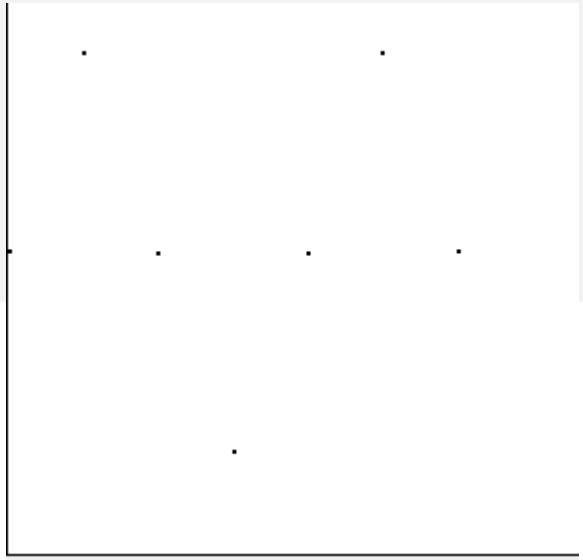
- Este bloque de código:
  - Importa la función SIN, y la constante PI.
  - Luego redimensiona el lienzo a 500 x 500 pixels
  - Finalmente redefine coordenadas.

Algunos Ejemplos Gráficos

Ing. Ventre, Luis O.

- Si creamos algunos puntos:

```
6  create_point(-2*pi, sin(-2*pi))
7  create_point(-1.5*pi, sin(-1.5*pi))
8  create_point(-pi, sin(-pi))
9  create_point(-0.5*pi, sin(-0.5*pi))
10 create_point(0, sin(0))
11 create_point(0.5*pi, sin(0.5*pi))
12 create_point(pi, sin(pi))
13 create_point(1.5*pi, sin(1.5*pi))
14 create_point(2*pi, sin(2*pi))
```



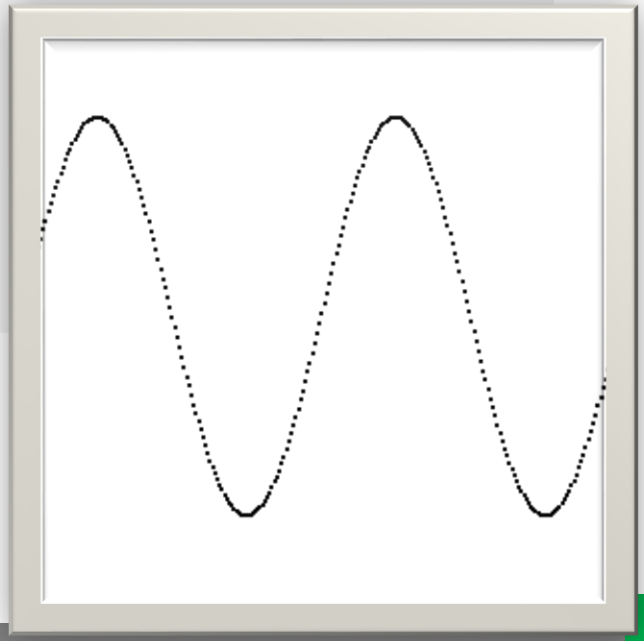
- Ahora es cuando debemos aplicar lo aprendido!!...para dibujar mas puntos, debemos utilizar un BUCLE.
- Podría ser un bucle WHILE, que inicie una variable en -2*PI. y vaya cambiando su valor sucesivamente.
- Hasta cuando?
- Hasta llegar al valor del otro extremo 2*PI.

- Ej.

seno_7.py

seno.py

```
1 from math import pi, sin
2
3 window_size (500, 500)
4 window_coordinates(-2*pi, -1.5, 2*pi, 1.5)
5
6 x = -2*pi
7 while x <= 2*pi:
8     create_point(x, sin(x))
9     x += 0.05
```

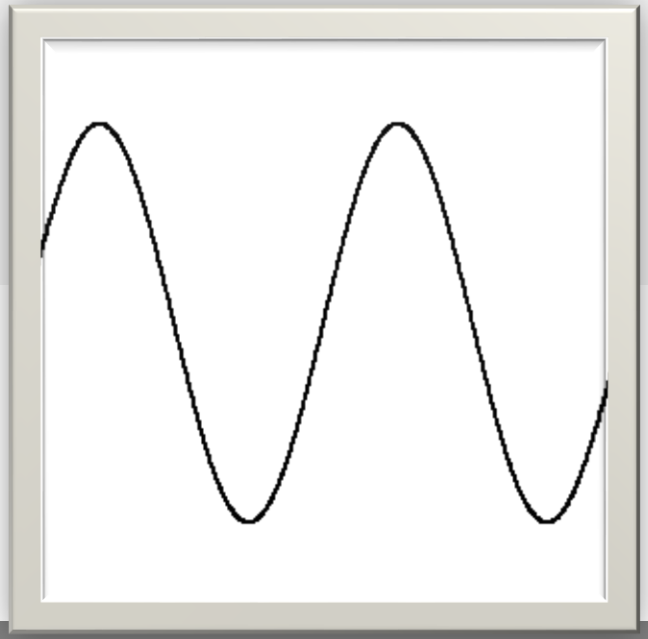


- Y si achicamos el intervalo?

seno_8.py

seno.py

```
1  from math import pi, sin
2
3  window_size (500, 500)
4  window_coordinates(-2*pi, -1.5, 2*pi, 1.5)
5
6  incremento = (2*pi - -2*pi) / 1000
7
8  x = -2*pi
9  while x <= 2*pi:
10     create_point(x, sin(x))
11     x += incremento
```



Algunos Ejemplos Gráficos

- Haz un ultimo intento, reemplaza la función create_point por create_line, puedes?

seno_10.py — seno.py

```
1  from math import pi, sin
2
3  x1 = float(raw_input('Dime el límite inferior del intervalo: '))
4  x2 = float(raw_input('Dime el límite superior del intervalo: '))
5  puntos = int(raw_input('Dime cuántos puntos he de mostrar: '))
6
7  window_size(500, 500)
8  window_coordinates(x1, -1.5, x2, 1.5)
9
10 incremento = (x2 - x1) / puntos
11
12 x = x1
13 while x <= x2 - incremento:
14     create_line(x, sin(x), x+incremento, sin(x+incremento))
15     x += incremento
```

- Analiza las posibilidades y compara resultados!.

Repasando!...

- **Sentencias de control ITERATIVAS**
- **La sentencia WHILE.**
- **Calculo de SUMATORIOS**
- **Programa de REQUISITOS en la Entrada…solicito tantas veces los datos hasta que se ingresen correctamente.**

- **Mejorando el programa MENUS!...como hacer para salir del programa con el ingreso de una opcion especifica.**
- **El bucle FOR-IN.**
- **La utilización de FOR-IN reemplazando WHILE.**
- **Números PRIMOS.**
- **Rotura de bucles - BREAK**

- **Anidamiento de estructuras FOR-IN adentro FOR-IN**
- **Captura y tratamiento de EXCEPCIONES> TRY-EXCEPT**
- **Graficador de FUNCIONES.**
- **Ver…restantes ejemplos de gráficos capitulo 4.**