

# **TEMA 2**

# **PROGRAMACIÓN**



# TEMA 2. PROGRAMACIÓN

- ❑ 2.1 Algoritmos
- ❑ 2.2 Tipos de datos simples y variables
- ❑ 2.3 Instrucciones básicas
- ❑ 2.4 Estructuras de control
- ❑ 2.5 Tipos de datos estructurados: Secuencias
- ❑ 2.6 Funciones

## 2.1 Algoritmos

Un algoritmo es una secuencia finita de **acciones primitivas**, entendiendo como tales, aquellas que pueden ser ejecutadas por el procesador.

### Técnicas para el diseño de algoritmos:

- Programación estructurada
  - ✂ Programa con diseño modular
  - ✂ Cada módulo tiene diseño descendente
  - ✂ Cada módulo se codifica con estructuras de control secuenciales, selectivas e iterativas

## 2.1 Algoritmos: Lenguaje Python

**Python** fue creado a principios de los noventa por Guido van Rossum en los Países Bajos. Es relativamente joven.

### Ventajas

- ❑ Python es un lenguaje **muy expresivo**, es decir, los programas Python son **muy compactos**. Un programa Python suele ser bastante más corto que su equivalente en lenguajes como C. Python llega a ser considerado por muchos un *lenguaje de programación de muy alto nivel*.
- ❑ Python es **muy legible**. La sintaxis de Python es muy elegante y permite la escritura de programas cuya lectura resulta más fácil que si utilizáramos otros lenguajes de programación.

## 2.1 Algoritmos: Python

### Ventajas

- ❑ Python ofrece un **entorno interactivo** que facilita la realización de pruebas y ayuda a despejar dudas acerca de ciertas características del lenguaje.
- ❑ El **entorno de ejecución** de Python *detecta muchos de los errores* de programación que escapan al control de los compiladores y proporciona información muy rica para detectarlos y corregirlos.
- ❑ Python puede usarse **como lenguaje imperativo procedimental** o como **lenguaje orientado a objetos**.
- ❑ Posee un **rico juego de estructuras de datos** que se pueden manipular de modo sencillo.

## 2.2 Tipos de datos simples y variables

### Programas o scripts

Los programas en Python tienen extensión .py

#### Ejemplos:

perimetro.py

```
1 from math import pi
2 radio = 1
3 perímetro = 2 * pi * radio
4 print(perímetro)
```

esfera.py

```
1 from math import pi
2
3 cadena_leída = input()
4 radio = float(cadena_leída)
5
6 volumen = 4 / 3 * pi * radio ** 3
7
8 print(volumen)
```

## 2.2 Tipos de datos simples

### ❑ Numéricos

- **Enteros** *int*, **reales** *float*, **complejos** *complex* y **lógicos** *boolean*.

### ❑ Cadenas de textos

- No existe el tipo carácter.
- **Cadena** *str* : secuencia de caracteres
  - Ejemplo: cadena='buenos días'

## 2.2 Tipos de datos simples y variables

### ■ Entero *int*

16 bits de representación.

#### ■ Operadores

- Binarios de relación: =, <, >, <>, <=, >=
- Aritméticos: +, -, \*, //, %, \*\*
- *Casting* (convertir el valor de una variable de un tipo a otro): *int()*

Ejemplos: 5 // 3 es 1

5 % 3 es 2

5 \*\* 2 es 25

int(-2.9) es -2

int('2') es 2



## 2.2 Tipos de datos simples y variables

### ❑ Real *float*

- ❑ Python utiliza doble precisión: 64 bits
- ❑ Operadores
  - ❑ Relacionales: `=`, `<`, `>`, `<>`, `<=`, `>=`
  - ❑ Aritméticos: `+`, `-`, `*`, `/`
  - ❑ Funciones:
    - ❑ Predefinidas: *abs*, *round*, *float*...
    - ❑ Definidas en módulos: *sin*, *cos* (módulo *math*)...

Ejemplo: *abs*(-3) es 3

*float*('3.2') es 3.2

## 2.2 Tipos de datos simples y variables

### ❑ Lógico *boolean*

- ❑ Un dato de tipo lógico sólo puede presentar uno de dos valores: ***True o False***
- ❑ Operadores
  - ❑ Relacionales: ==, <, >, !=, <=, >=
  - ❑ Lógicos: **and, or, not**

Ejemplos: 3!=5 es True

3<5 and 17<10 es False

## 2.2 Tipos de datos simples y variables

### ❑ Lógico *boolean*

- Un dato de tipo lógico sólo puede presentar uno de dos valores: *True* o *False*
- *Subtipo de enteros*

Operation	Result
<code>x or y</code>	if <code>x</code> is false, then <code>y</code> , else <code>x</code>
<code>x and y</code>	if <code>x</code> is false, then <code>x</code> , else <code>y</code>
<code>not x</code>	if <code>x</code> is false, then <code>True</code> , else <code>False</code>

## 2.2 Tipos de datos simples y variables

### **Cualquiera de estos valores se considera False**

None

False

Cero de cualquier tipo, 0, 0L, 0.0, 0j.

Una secuencia sin elementos, "", (), [].

Un diccionario vacío, {}.

## 2.2 Tipos de datos simples y variables

Utilizaremos **variables en nuestros programas para guardar valores**. Estas variables serán de los tipos que hemos descrito.

- El nombre de una variable es su **identificador**: formado por letras minúsculas, mayúsculas, dígitos y/o el carácter de subrayado, con una restricción: que el primer carácter no sea un dígito.
- Este identificador no debe coincidir con ninguna palabra reservada: *and, as, assert, break, class, continue, def, del, elif, else, except, False, finally, for, from, global, if, import, in, is, lambda, nonlocal, None, not, or, pass, raise, return, True, try, with, while* y *yield*.
- Python distingue entre mayúsculas y minúsculas, así que *área*, *Area* y *AREA* son tres identificadores válidos y diferentes.

## 2.3 Instrucciones básicas

### □ Asignación

En Python, la primera operación sobre una variable debe ser la asignación de un valor.

**variable=expresion**

Se evalúa la expresión a la derecha del símbolo igual y se guarda el valor resultante en la variable indicada a la izquierda del símbolo igual.

- Recuerda que comparar `==` no es asignar `=`

**Asignaciones con operador:** `+=`, `-=`, `*=`, `/=`, `%=`, `//=`, `**=`

Ejemplos:

```
>>> x = 3↵
>>> x = x + 1↵
>>> x↵
4
```

```
>>> a = 3↵
>>> b = 2↵
>>> a += 4 * b↵
>>> a↵
11
```





## 2.3 Funciones

**Functions are like their own little programs. They take input, which we call the function arguments (or parameters) and give us back output that we refer to as return values.**

```
>>> x = 3.5  
>>> abs(x)
```



## 2.3 Tipos de datos

Function	What it does	Example
<code>type(<i>n</i>)</code>	Get the type of <i>n</i>	<code>type(13)</code>  <code>int</code>
<code>int(<i>n</i>)</code>	Convert <i>n</i> to type <b>int</b>	<code>int("45")</code>  <code>45</code>
<code>float(<i>n</i>)</code>	Convert <i>n</i> to type <b>float</b>	<code>float(45)</code>  <code>45.0</code>
<code>str(<i>n</i>)</code>	Convert <i>n</i> to type <b>str</b>	<code>str(4.0)</code>  <code>'4.0'</code>



## 2.3 Métodos

**Methods are functions that are called using the dot notation.**

```
an_object.an_attribute
```

```
an_object.a_method()
```

**Use function `dir()` to display available methods and attributes**

```
>>> a = 2.5
```

```
>>> dir(a)
```

## 2.3 Entrada-salida



## 2.3 Instrucciones básicas

### ❑ Lectura

Vamos a aprender a hacer que nuestro programa, cuando se ejecute, pida datos que se introduzcan desde teclado: utilizaremos ***input()***

Esta función detiene la ejecución del programa y espera a que el usuario escriba un texto y pulse la tecla de retorno de carro; en ese momento prosigue la ejecución y la función devuelve una cadena con el texto que tecleó el usuario.

```
nombre = input("Como te llamas? ")
```

```
num = eval(input("Introduce un numero: "))
```

## 2.3 Instrucciones básicas

### ❑ Escritura

***print(expresion, [expresion,])***

Formato de escritura ***format***

Interpolar valores en una cadena:

“la suma de {} y {} es {}".format(4,5,4+5)

“la suma de {0} y {1} es {2}".format(4,5,4+5)

“la suma de {uno} y {dos} es {suma}".format(unos=4,dos=5,suma=4+5)

Para mostrar los valores en coma flotante con un solo decimal

“{} entre {} es {:.1f}".format(10,3,10/3)

## 2.3 Tipos de datos: Cadenas de texto

### ❑ Cadenas ***str***

- secuencia de caracteres
- saludo ='buenos días'

Python ofrece una serie de operadores y funciones predefinidos para las secuencias de caracteres:

- ❑ Operador + : acepta dos cadenas como operandos y devuelve la cadena que resulta de unir la segunda a la primera.
- ❑ Operador \*: acepta una cadena y un entero y devuelve la concatenación de la cadena consigo misma tantas veces como indica el entero.
- ❑ *len*: devuelve la longitud de una cadena

## 2.3 Tipos de datos: Cadenas de texto

### ❑ Cadenas ***str***

Python ofrece una serie de operadores y funciones predefinidos:

- ❑ *str*: se le pasa un objeto y devuelve una representación del valor como secuencia de caracteres.
- ❑ *int*: recibe una cadena cuyo contenido es una secuencia de dígitos y devuelve el número entero que describe.
- ❑ *float*: acepta una cadena cuyo contenido describe un flotante y devuelve el flotante en cuestión.

## 2.3 Tipos de datos: Cadenas de texto

### ❑ Cadenas *str*

Podemos también manipular cadenas mediante **métodos** que les son propios:

- ❑ *a.lower()* (paso a minúsculas): devuelve una cadena con los caracteres de *a* convertidos en minúsculas.
- ❑ *a.upper()* (paso a mayúsculas): devuelve una cadena con los caracteres de *a* convertidos en mayúsculas.
- ❑ *a.format(expr1 expr2 . . . )* (sustitución de marcas de formato): devuelve una cadena en la que las marcas de formato de *a* se sustituyen por el resultado de evaluar las expresiones dadas.
- ❑ *a.split()* : obtiene una lista con todas las palabras de una cadena.

```
>>> 'uno_dos_tres'.split()↵  
['uno', 'dos', 'tres']
```

```
>>> ' _uno_dos_tres_'.split()↵  
['uno', 'dos', 'tres']
```

## 2.3 ASCII code

*ord*: acepta una cadena compuesta por un único carácter y devuelve su código Unicode (un entero).

*chr*: recibe un entero y devuelve una cadena con el carácter que tiene a dicho entero como código Unicode.

```
>>> ord('A')
```

```
65
```

```
>>> chr(32)
```

```
' '
```

```
>>> chr(65)
```

```
'A'
```

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	\$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□



## 2.3 Tipos de datos: Cadenas de texto

### ■ Cadenas ***str***

Es posible incluir ciertos caracteres especiales que no tienen una representación trivial:

Salto de línea `\n`

`\t` tabulador horizontal

`\\` barra invertida

`'` comilla simple

`''` comilla doble

`\` y salto de línea: para expresar una cadena en varias líneas

## 2.3 Usando módulos

**En Python podemos guardar nuestras definiciones de variables (y funciones como ya veremos) en un archivo al que llamamos módulo**

**Dichas definiciones se pueden importar en nuestros programas**

```
>>> import math
```

**dir() nos muestra las funciones y otros objetos disponibles:**

```
>>> dir(math)
```

**Y podemos usar help() para tener ayuda sobre una función**

```
>>> help(math.cos)
```

## 2.3 Usando módulos: math

Este módulo siempre está disponible y permite acceder a funciones matemáticas de C

Estas funciones no están disponibles para números complejos, para los que existe el módulo ***cmath***

---

<code>acos(x)</code>	returns the arccosine of $x$ ( $\cos^{-1} x$ )
<code>asin(x)</code>	returns the arcsine of $x$ ( $\sin^{-1} x$ )
<code>atan(x)</code>	returns the arctangent of $x$ ( $\tan^{-1} x$ )
<code>atan2(y, x)</code>	returns the arctangent of $y/x$ ( $\tan^{-1}(y/x)$ )
<code>cos(x)</code>	returns the cosine of $x$ radians ( $\cos x$ )
<code>degrees(x)</code>	returns the number of degrees in $x$ radians
<code>exp(x)</code>	returns $e^x$
<code>log(x, [b])</code>	returns the logarithm base $b$ of $x$ ( $\log_b x$ ); if $b$ is omitted, returns the natural logarithm of $x$ ( $\ln x$ )
<code>radians(x)</code>	returns the number of radians in $x$ degrees
<code>sin(x)</code>	returns the sine of $x$ radians ( $\sin x$ )
<code>sqrt(x)</code>	returns the square root of $x$ ( $\sqrt{x}$ )
<code>tan(x)</code>	returns the tangent of $x$ radians ( $\tan x$ )
<code>e</code>	the value of $e$ (Euler's number), the base of the natural logarithm
<code>pi</code>	the value of $\pi$

---

## 2.3 Programas

Los programas de Python tienen extensión .py

Podemos ejecutarlos invocando al interprete de python junto al nombre del fichero

```
% python mi_programa.py
```

También podemos definir módulos (mi\_modulo.py) e importarlos

```
import modulo
```

## 2.3 Programa

```
#!/usr/bin/python
```

```
#
```

```
# Let's calculate diameter, circumference and area for a circle of given radius
```

```
#
```

```
import math
```

```
r = input("What is the radius of your circle (in cm)? ")
```

```
print("The diameter of your circle is {} cm".format(2*r))
```

```
print("The circumference of your circle is {} cm".format(2*math.pi*r))
```

```
print("The area of your circle is {} cm2".format(math.pi*pow(r,2)))
```

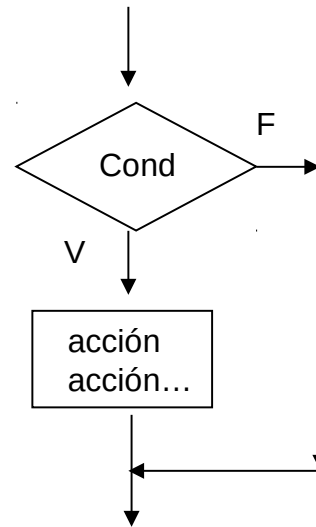
## 2.4 Estructuras de control

- ❑ Sentencias condicionales
  - ❑ Sentencia **if**
  - ❑ Sentencia **if-else**
  - ❑ Forma compacta **elif**
- ❑ Sentencias iterativas
  - ❑ Sentencia **while**
  - ❑ Sentencia **for-in**

## 2.4 Sentencias condicionales

### ■ Sentencia **if**

```
if condición:  
    acción  
    acción  
    ...  
    acción
```



## 2.4 Sentencias condicionales

### ❑ Sentencia **if**

Ejemplo:

```
primer_grado.py
1 print('Programa para la resolución de la ecuación  $ax + b = 0$ .')
2
3 a = float(input('Valor de a: '))
4 b = float(input('Valor de b: '))
5
6 if a != 0:
7     x = -b / a
8     print('Solución: ', x)
9
```



## 2.4 Sentencias condicionales

### ❑ Sentencia **if**

misterio.py

```
1 letra = input('Dame una letra minúscula: ')
2
3 if letra <= 'k':
4     print('Es de las primeras del alfabeto')
5 if letra >= 'l':
6     print('Es de las últimas del alfabeto')
```

## 2.4 Sentencias condicionales

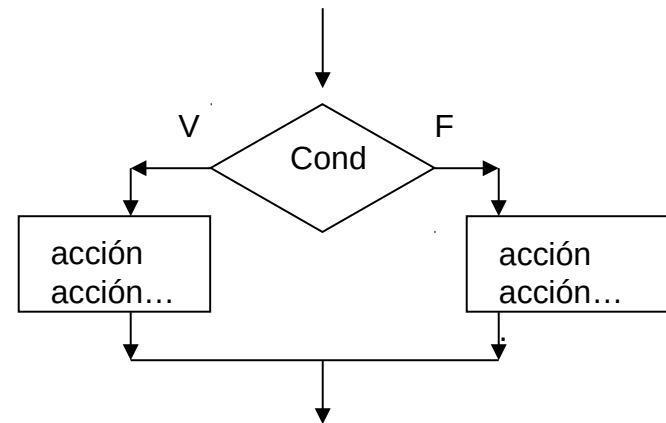
- Se pueden establecer sentencias **if** anidadas

```
primer_grado.py
1 print('Programa para la resolución de la ecuación  $ax + b = 0$ .')
2
3 a = float(input('Valor de a: '))
4 b = float(input('Valor de b: '))
5
6 if a != 0:
7     x = -b / a
8     print('Solución: ', x)
9
10 if a == 0:
11     if b != 0:
12         print('La ecuación no tiene solución.')
13     if b == 0:
14         print('La ecuación tiene infinitas soluciones.')
```

## 2.4 Sentencias condicionales

### ■ Sentencia **if-else**

```
if condición:  
    acciones  
else:  
    otras acciones
```



## 2.4 Sentencias condicionales

### ■ Sentencia if-else

```
segundo_grado.py
1 from math import sqrt # La función sqrt calcula la raíz cuadrada de un número.
2
3 print('Programa para la resolución de la ecuación  $ax^2+bx+c=0$ .')
4
5 a = float(input('Valor de a: '))
6 b = float(input('Valor de b: '))
7 c = float(input('Valor de c: '))
8
9 if a != 0:
10     x1 = (-b + sqrt(b**2 - 4*a*c)) / (2 * a)
11     x2 = (-b - sqrt(b**2 - 4*a*c)) / (2 * a)
12     print('Soluciones: x1={0:.3f} y x2={1:.3f}'.format(x1, x2))
13 else:
14     print('No es una ecuación de segundo grado.')
```

## 2.4 Sentencias condicionales

### Sentencia if-else

```
segundo_grado.py
1 from math import sqrt # La función sqrt calcula la raíz cuadrada de un número.
2
3 print('Programa para la resolución de la ecuación  $ax^2+bx+c=0$ .')
4
5 a = float(input('Valor de a: '))
6 b = float(input('Valor de b: '))
7 c = float(input('Valor de c: '))
8
9 if a != 0:
10     x1 = (-b + sqrt(b**2 - 4*a*c)) / (2 * a)
11     x2 = (-b - sqrt(b**2 - 4*a*c)) / (2 * a)
12     print('Soluciones: x1={0:.3f} y x2={1:.3f}'.format(x1, x2))
13 else:
14     if b != 0:
15         x = -c / b
16         print('Solución: x={0:.3f}'.format(x))
17     else:
18         if c != 0:
19             print('La ecuación no tiene solución.')
20         else:
21             print('La ecuación tiene infinitas soluciones.')
```

## 2.4 Sentencias condicionales

- Forma compacta **elif**

```
if condición:  
    ...  
else:  
    if otra condición:  
        ...
```



```
if condición:  
    ...  
elif otra condición:  
    ...
```

## 2.4 Sentencias condicionales

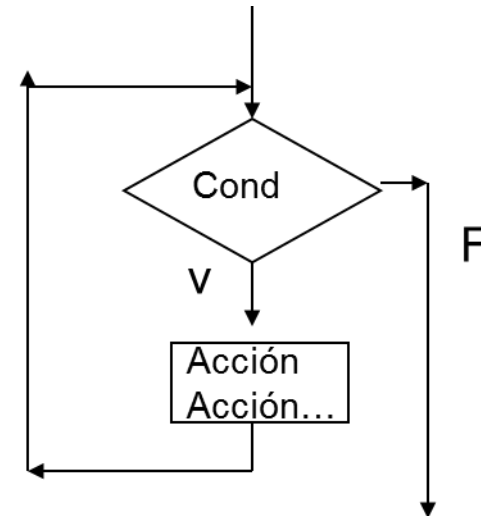
### □ Forma compacta **elif**

```
circulo.py
1 from math import pi
2
3 radio = float(input('Dame el radio de un círculo: '))
4
5 # Menú
6 print('Escoge una opción: ')
7 print('a) Calcular el diámetro.')
8 print('b) Calcular el perímetro.')
9 print('c) Calcular el área.')
10 opción = input('Teclea a, b o c y pulsa el retorno de carro: ')
11
12 if opción == 'a': # Cálculo del diámetro.
13     diámetro = 2 * radio
14     print('El diámetro es {0}'.format(diámetro))
15 elif opción == 'b': # Cálculo del perímetro.
16     perímetro = 2 * pi * radio
17     print('El perímetro es {0}'.format(perímetro))
18 elif opción == 'c': # Cálculo del área.
19     área = pi * radio ** 2
20     print('El área es {0}'.format(área))
21 else:
22     print('Solo hay tres opciones: a, b o c.')
23     print('Tú has tecleado "{0}"'.format(opción))
```

## 2.4 Sentencias iterativas

### □ Sentencia **while**

```
while condición:  
    acción  
    acción  
    ...  
    acción
```





## 2.4 Sentencias iterativas

### ■ Sentencia **while**

```
contador.py
1 i = 0
2 while i < 3:
3     print(i)
4     i += 1
5 print('Hecho')
```

- Ten cuidado con los bucles infinitos:

```
⚡ bucle_infinito.py
1 i = 0
2 while i < 10:
3     print(i)
```

## 2.4 Sentencias iterativas

- Ejemplo: un programa que calcula la suma de los 1000 primeros números

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

## 2.4 Sentencias iterativas

### ❑ Rotura de bucles: **break**

```
es_primo.py
1 número = int(input('Dame un número: '))
2
3 if número > 1:
4     creo_que_es_primo = True
5     divisor = 2
6     while divisor < número:
7         if número % divisor == 0:
8             creo_que_es_primo = False
9             break
10        divisor += 1
11 else:
12     creo_que_es_primo = False
13
14 if creo_que_es_primo:
15     print('El número {0} es primo.'.format(número))
16 else:
17     print('El número {0} no es primo.'.format(número))
```

## 2.4 Generando números aleatorios

Los números aleatorios generados por ordenador no son realmente aleatorios sino pseudoaleatorios ya que están generados por una fórmula

Son muy útiles en simulaciones y juegos

Usamos el módulo *random*

```
import random
```

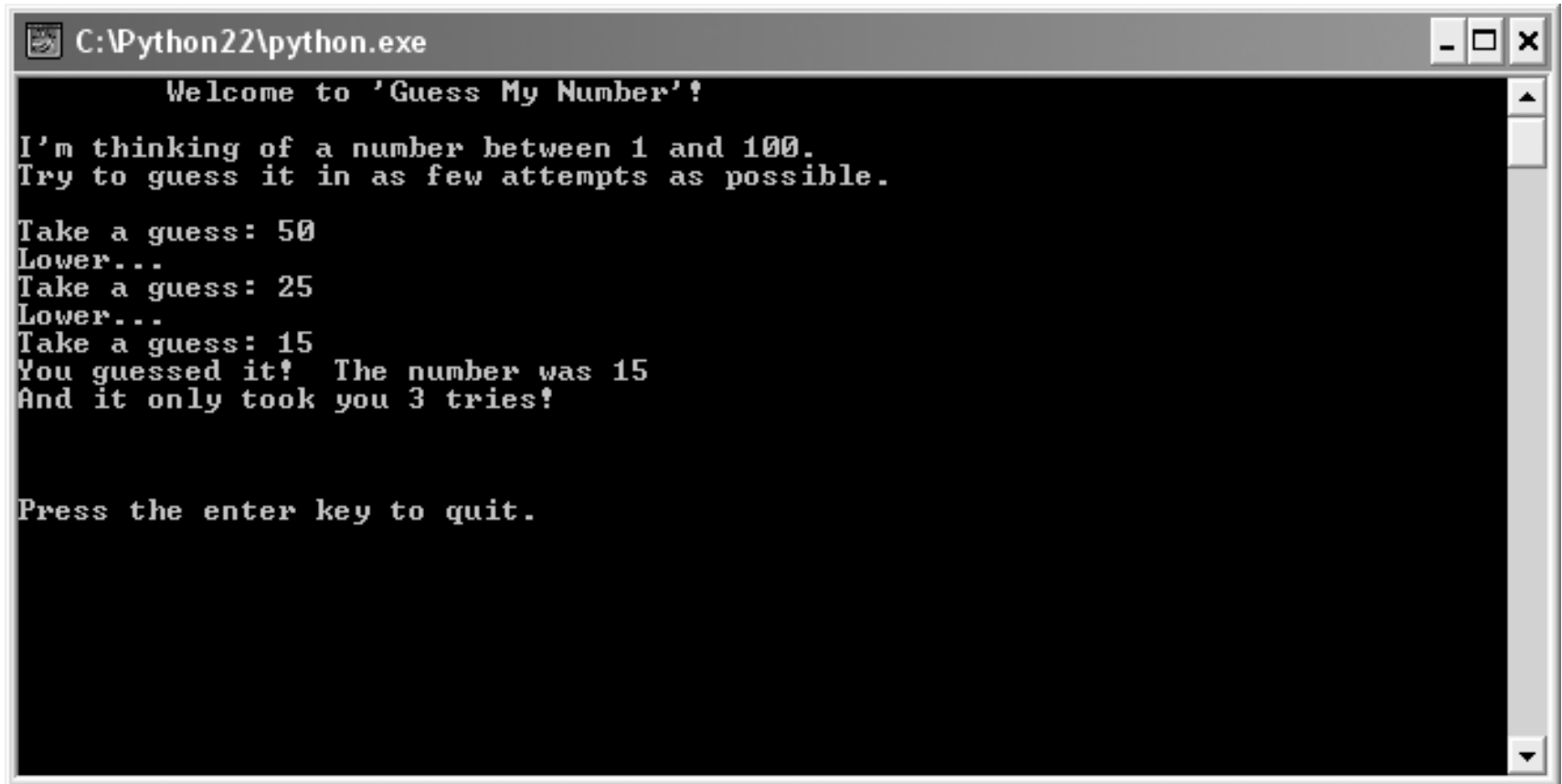
```
random.randrange(n) #generates random number from 0 to n - 1
```

## 2.4 random.randrange() function

`randrange(n)` returns random number from 0 to  $n - 1$

`randrange(n,m)` returns random number from  $n$  to  $m - 1$

## 2.4 Adivina el número



A screenshot of a Windows command prompt window titled "C:\Python22\python.exe". The window has a black background with white text. The text inside the window reads: "Welcome to 'Guess My Number'!", "I'm thinking of a number between 1 and 100.", "Try to guess it in as few attempts as possible.", "Take a guess: 50", "Lower...", "Take a guess: 25", "Lower...", "Take a guess: 15", "You guessed it! The number was 15", "And it only took you 3 tries!", and "Press the enter key to quit.". The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
C:\Python22\python.exe

Welcome to 'Guess My Number'!

I'm thinking of a number between 1 and 100.
Try to guess it in as few attempts as possible.

Take a guess: 50
Lower...
Take a guess: 25
Lower...
Take a guess: 15
You guessed it! The number was 15
And it only took you 3 tries!

Press the enter key to quit.
```

## 2.4 Adivina el número

```
from random import randrange
print("Bienvenida al juego!")

print("Estoy pensando en un número entre el 1 y el 100")
print("Intenta adivinarlo en el menor número de intentos")

c=0
n=randrange(1,101)
while True:
    a = input("Cuál es el número?: ")
    c+=1
    if (a>n):
        print("Menor...")
    elif(a<n):
        print("Mayor...")
    else:
        print("Lo has adivinado!")
        print("sólo has necesitado {} intentos".format(c))
        break
```

## 2.4 Sentencias iterativas

### ❑ Sentencia **for-in**

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



## 2.4 Sentencias iterativas

### ■ Función **range()**

*range(3)* es [0, 1, 2]

*range(2, 10)* es [2, 3, 4, 5, 6, 7, 8, 9]

*range(-3, 3)* es [-3, -2, -1, 0, 1, 2]

*range(-10, 0)* es [-10, -9, -8, -7, -6, -5, -4, -3, -2, -1]

*range(2, 10, 2)* es [2, 4, 6, 8]

*range(2, 10, 3)* es [2, 5, 8]

*range(10, 5, -1)* es [10, 9, 8, 7, 6]

```
for i in  
    range(10):  
        print(i)
```

## 2.4 Sentencias iterativas

### ❑ **for-in** como forma compacta de ciertos **while**

Ciertos bucles se ejecutan un número de veces fijo y conocido a priori:

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

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

## 2.4 Sentencias iterativas

### ❑ Índice del bucle **for-in**

En un bucle for-in, las variables de índice solo deben usarse para consultar su valor, nunca para asignarles uno nuevo.

Ejemplo: Es incorrecto:

```
1 for i in range(0, 5):  
2     i += 2
```

O utilizar el mismo índice para dos bucles anidados:

```
1 for i in range(0, 5):  
2     for i in range(0, 3):  
3         print(i)
```

## 2.2 Tipos de datos compuestos

**Cadena *str*** : secuencia de caracteres

Ejemplo: cadena='buenos días'

**Lista *list***: secuencia de elementos, potencialmente de distinto tipo, a la que se le puede eliminar elementos, añadir nuevos, y modificar valores individuales.

Ejemplo: lista=['hola',4, (1,2),{3,4}]

## 2.5 Tipos de datos estructurados: Secuencias

```
nombre = "ALBERTO"  
for letra in nombre:  
    print(letra)
```

## 2.5 Tipos de datos estructurados: Secuencias

### ■ Cadenas *str*

Podemos acceder a cada uno de los caracteres de una cadena utilizando un operador de indexación:

`a[i]` es el carácter que ocupa la posición `i+1` en `a`

```
>>> a =  
"ABCDEFGH"  
>>> a[0]  
'A'  
>>> a[1]  
'B'  
>>> a[-1]  
'H'  
>>> a[-2]  
'G'
```

## 2.5 Tipos de datos estructurados: Secuencias

- ❑ **Subcadenas**

- ❑ La expresión `a[i:j]` significa que se desea obtener la subcadena formada por los caracteres `a[i]` , `a[i+1]` , . . . , `a[j-1]`.

```
>>> a[2:4]
'CD'
```

```
>>> a[:3]
'ABC'
```

- ❑ El corte `a[:j]` es equivalente a `a[0,j]` y el corte `a[i:]` equivale a `a[i:len(a)]`

```
>>> a[2:]
'CDEFG'
```

- ❑ `a[0:len(a):2]` selecciona los caracteres de índice par

```
>>> a[0:len(a):2]
'ACEG'
```

- ❑ Invertir la cadena: `a[::-1]`

```
>>> a[::-1]
'GFEDCBA'
```

## 2.5 Tipos de datos estructurados: Secuencias

### ❑ Cadenas *str*

- **Cuidado:** Las cadenas son inmutables



```
>>> a = "ABCD"
```

```
>>> a[1]  
'B'
```

```
>>> a[1] = "C"
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'str' object does not support item assignment



## 2.5 Tipos de datos estructurados: Secuencias

### ■ Listas *list*

Python nos permite definir secuencias de valores de cualquier tipo. Los valores de una lista deben estar entre corchetes y separados por comas.

```
contactos = ['Maria','Juan','Miguel', 'Sara']  
  
print('Tengo {} amigos'.format(len(contactos)))  
for nombre in contactos:  
    print(nombre)
```

## 2.5 Tipos de datos estructurados: Secuencias

### □ Listas *list*

Muchos de los operadores y funciones que trabajan sobre cadenas, también lo hacen sobre listas: len, +, \*, operador de corte:

```
>>> a = [1, 2, 3]
```

```
>>> b = [4, 5, 6]
```

```
>>> len(a)
```

```
3
```

```
>>> a + b
```

```
[1, 2, 3, 4, 5, 6]
```

```
>>> a * 2
```

```
[1, 2, 3, 1, 2, 3]
```

## 2.5 Tipos de datos estructurados: Secuencias

### □ Listas *list*

*Modificación de elementos de listas:*

```
>>> lista = [1, 2, 3]
```

```
>>> lista
```

```
[1, 2, 3]
```

```
>>> lista[1] = 4
```

```
>>> lista
```

```
[1, 4, 3]
```

## 2.5 Tipos de datos estructurados: Secuencias

### □ **append(), del(), pop()**

```
>>> a = [1, 2, 3]
```

```
>>> a + [4]
```

```
[1, 2, 3, 4]
```

```
>>> a.append(4)
```

```
>>> a
```

```
[1, 2, 3, 4]
```

```
>>> del(a[1])
```

```
>>> a
```

```
[1, 3, 4]
```

```
>>> a.pop()
```

```
4
```

```
>>> a
```

```
[1, 3]
```

```
>>> a.pop(0)
```

```
1
```

## 2.5 Tipos de datos estructurados: Secuencias

### □ Listas *list*

Mutabilidad, inmutabilidad y representación de la información en memoria:

- Python procura almacenar en memoria una sola vez cada valor inmutable

```
>>> nombre1 = "Juan"
>>> nombre2 = "Juan"
>>> nombre1 ==
nombre2
True
>>> nombre1 is nombre2
True
```

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a == b
True
>>> a is b
False
>>> c = a
>>> a is c
True
```

## 2.5 Tipos de datos estructurados: Secuencias

### □ Leyendo listas

```
>>> n = 5
>>> a = []
>>> for i in range(n):
...     el = input('Elemento:
...')
...     a.append(el)
...
Elemento: 1
Elemento: 2
Elemento: 3
Elemento: 4
Elemento: 5
>>> a
[1, 2, 3, 4, 5]
```

```
lista: [1,2,3,4,5]
>>> eval(lista)
[1, 2, 3, 4, 5]
```

## 2.5 Tipos de datos estructurados: Secuencias

### □ Listas *list*

- *split()*
- *join()*

```
>>> lista = input("lista: ")  
lista: A B C D E F
```

```
>>> lista  
'A B C D E F'
```

```
>>> lista2 = lista.split()  
>>> lista2  
['A', 'B', 'C', 'D', 'E', 'F']
```

```
>>> ','.join(lista2)  
'A,B,C,D,E,F'
```

## 2.5 Tipos de datos estructurados: Secuencias

```
# Reading an mxn matrix
m = input("Number of rows?" )
n = input("Number of columns?" )
M = [[None] * n] * m
for i in range(m):
    for i in range(n):
        M[i,j]= input("Element ({},{})".format(i,j))
```

```
>>> M = input('M= ')
M= [[1,2,3],[4,5,6],[7,8,9]]
```

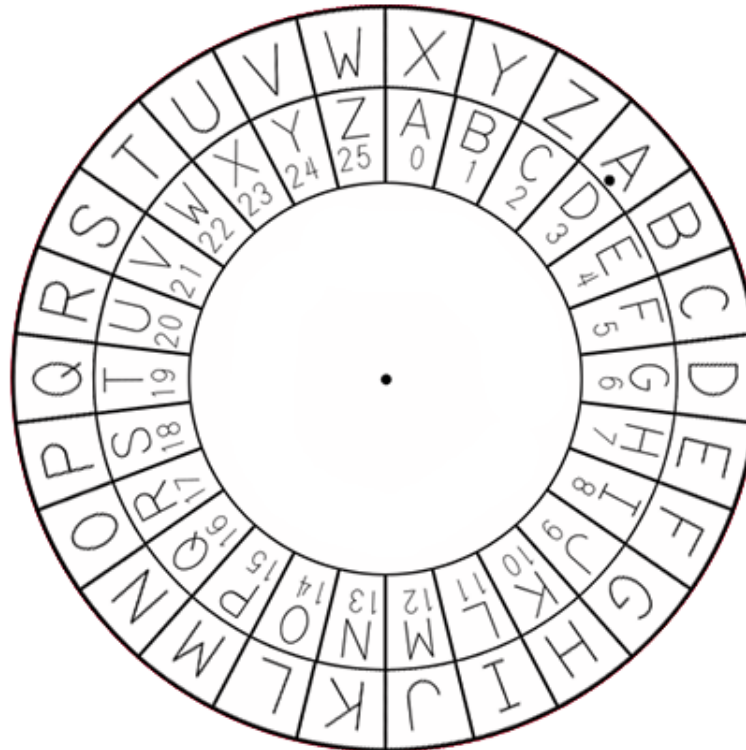
```
>>> eval(M)
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```



## 2.2 Tipos de datos compuestos

- ❑ **Conjunto *set*:** colección de elementos (sin orden y no repetidos), potencialmente de distinto tipo simple al que se le pueden añadir nuevos elementos o eliminar existentes.
  - Ejemplo: conj1={'infinito',1,0,5,('a',1)}
- ❑ **Conjunto congelado *frozenset* :** colección de elementos, potencialmente de distinto tipo (mientras estén entre los simples e inmutables), no repetidos y sin orden entre sí
  - Ejemplo: congelado=frozenset({3,5,6.1})
- ❑ **Tupla *tuple*:** secuencia de 0, 1 o n elementos, potencialmente de distinto tipo  
Ejemplo: t1=(1,'a',3,3)
- ❑ **Diccionario *dict*:** colección (conjunto) de pares de clave-valor. La clave es de cualquier tipo inmutable. Los valores pueden ser de cualquier tipo.  
Ejemplo: ingredientes={'tomate':(1,'Kg'), 'pepino':2,'sal':1 cucharilla, 'aceite':1, 'oregano':1 pizca'}

# Cifrado César



Click wheel to rotate.

X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

# Algoritmo

1. Decide una clave de 1 a 25.
2. Busca el número de la letra
3. Súmale la clave  $x$
4. Si el numero es mayor de 26, réstale 26.
5. Busca la letra corespondiente al nuevo número.
6. Repite los pasos anteriores hasta encontrar la nueva letra