UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TEMA 3

2021/2022

Programación

Tema 3. Estructuras de selección o decision

Índice

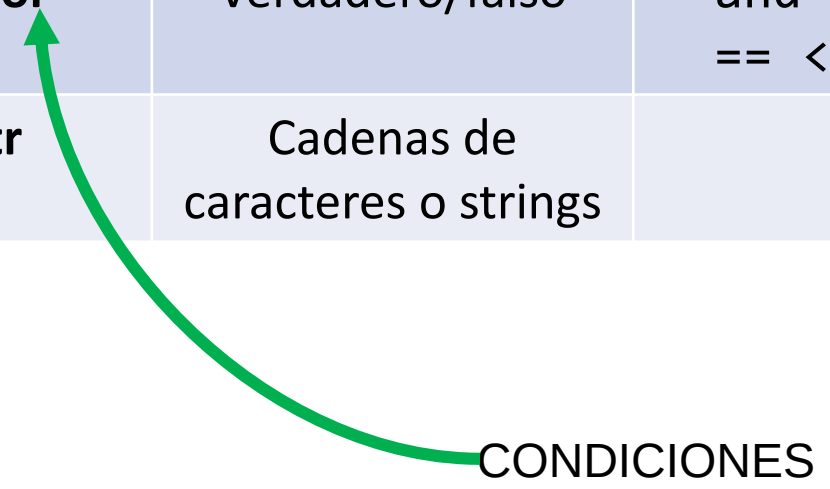
TEMA 3

1. Introducción

- Para resolver problemas reales casi siempre hay que tomar decisiones en función de les circunstancias y, entonces, escoger una de entre varias alternativas.
- Hasta ahora hemos escrito código que se ajusta a un flujo de control del programa **secuencial**.
- Ahora vemos instrucciones que permitan alterar este flujo secuencial en función de los datos de entrada o de resultados intermedios que se generen en la ejecución.
- Las estructuras de control que permiten cambiar el flujo de ejecución de un programa son la **selección** y la **repetición** (o iteración).
- Para saber cuando cambiar el flujo necesitamos CONDICIONES

1. Tipos de datos

Tipo de dato	Conjunto de valores	Operadores básicos	Ejemplos
int	Números enteros	`+ - * // % **`	99 12 2147483647
float	Números reales	`+ - * / **`	3.14 2.5 6.022e23
bool	Verdadero/falso	`and or not == <= >= > <`	True False
str	Cadenas de caracteres o strings	`+`	'AB' 'Hola' '2.5' “Hello” “where?”



1. Operadores lógicos

- El tipo de dato booleano (*bool*) representa valores de verdad:
  - verdadero (*True*) o
  - falso (*False*)
- Los operadores definidos para bool son **operadores lógicos**:
  - `a and b` es
    - `True` si ambos operandos son `True`, y
    - `False` si alguno de los dos es `False`
  - `a or b` es
    - `False` si ambos operandos son `False`, y
    - `True` si alguno de los dos es `True`
  - `not a` es
    - `True` si a es `False`, y
      - `False` si a es `True`

a	not a
False	True
True	False

a	b	a and b	a or b
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

1. Operadores lógicos

- **not** tiene mayor precedencia que el **and**
- **and** tiene mayor precedencia que el **or**

- Usar paréntesis con los operadores lógicos para expresiones más complejas
- A menudo, la misma función puede formularse de distintas formas. Por ejemplo, la expresión **(a and b)** y **not (not a or not b)** son equivalentes, como se demuestra en la siguiente tabla de verdad:

a	b	a and b	not a	not b	not a or not b	not (not a or not b)
False	False	False	True	True	True	False
False	True	False	True	False	True	False
True	False	False	False	True	True	False
True	True	True	False	False	False	True

1. Operadores relacionales

- Operadores relacionales toman operandos de un tipo y producen un booleano
- Los más importantes para números enteros y reales son:

Operador	Significado	True	False
==	Igual	2 == (1 + 1)	2.0 == 3.5
!=	Distinto	5.0 != 2.5	(2 * 2) != 4
<	Menor	2.5 < 5.0	5 < 5
<=	Menor o igual	5 <= 5	5.0 <= 2.5
>	Mayor	5.0 > 2.5	5 > 5
>=	Mayor o igual	5 >= 5	2.5 >= 5.0

1. Operadores relacionales

- Los operadores relacionales tienen menor precedencia que los aritméticos, y mayor que los lógicos

  - No hace falta poner paréntesis en expresiones como:

```
(2.0*2.0 - 4.0*3.0*5.0) >= 0.0
(5 >= 1) and (5 <= 12)
```

  - Aunque por estilo son recomendables

1. Operadores relacionales para strings

- Los operadores **==** y **!=** permiten comparar si dos strings son iguales o distintas

- Con los comparadores <, <=, > y >= se utiliza el código ASCII para comparar.

- ASCII = *American Standard Code for Information Interchange*

- Cada carácter tiene un código numérico ASCII

- Python usa este código para comparar
  - 'abajo' es menor que 'arriba',
  - 'Barco' es menor que ambos y 'ábaco' es mayor

Codigo ASCII

acrónimo inglés de *American Standard Code for Information Interchange*

Binario	Dec	Hex	Representación
0010 0000	32	20	espacio ( )
0010 0001	33	21	!
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29	)
0010 1010	42	2A	*
0010 1011	43	2B	+
0010 1100	44	2C	,
0010 1101	45	2D	-
0010 1110	46	2E	.
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6
0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9
0011 1010	58	3A	:
0011 1011	59	3B	;
0011 1100	60	3C	<
0011 1101	61	3D	=
0011 1110	62	3E	>
0011 1111	63	3F	?

Binario	Dec	Hex	Representación
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O
0101 0000	80	50	P
0101 0001	81	51	Q
0101 0010	82	52	R
0101 0011	83	53	S
0101 0100	84	54	T
0101 0101	85	55	U
0101 0110	86	56	V
0101 0111	87	57	W
0101 1000	88	58	X
0101 1001	89	59	Y
0101 1010	90	5A	Z
0101 1011	91	5B	[
0101 1100	92	5C	\
0101 1101	93	5D	]
0101 1110	94	5E	^
0101 1111	95	5F	_

Binario	Dec	Hex	Representación
0110 0000	96	60	`
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f
0110 0111	103	67	g
0110 1000	104	68	h
0110 1001	105	69	i
0110 1010	106	6A	j
0110 1011	107	6B	k
0110 1100	108	6C	l
0110 1101	109	6D	m
0110 1110	110	6E	n
0110 1111	111	6F	o
0111 0000	112	70	p
0111 0001	113	71	q
0111 0010	114	72	r
0111 0011	115	73	s
0111 0100	116	74	t
0111 0101	117	75	u
0111 0110	118	76	v
0111 0111	119	77	w
0111 1000	120	78	x
0111 1001	121	79	y
0111 1010	122	7A	z
0111 1011	123	7B	{
0111 1100	124	7C	|
0111 1101	125	7D	}
0111 1110	126	7E	~

ord

```
>>> ord("b")
98
>>> ord("B")
66
>>> ord("a")
97
>>> ord("á")
225
>>> ord("n")
110
>>> ord("ñ")
241
```

chr

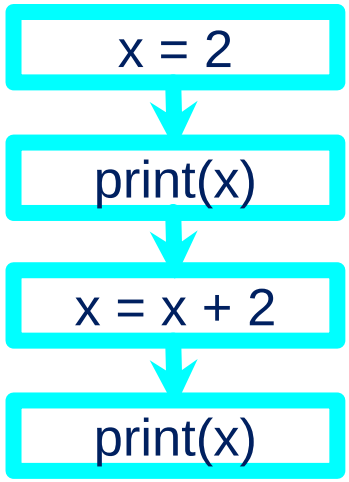
```
>>> chr(98)
'b'
>>> chr(66)
'B'
>>> chr(97)
'a'
>>> chr(225)
'á'
>>> chr(110)
'n'
>>> chr(241)
'ñ'
```

ejercicios reader 1 - 5

TEMA 3

Índice

2. Secuencial



Program:

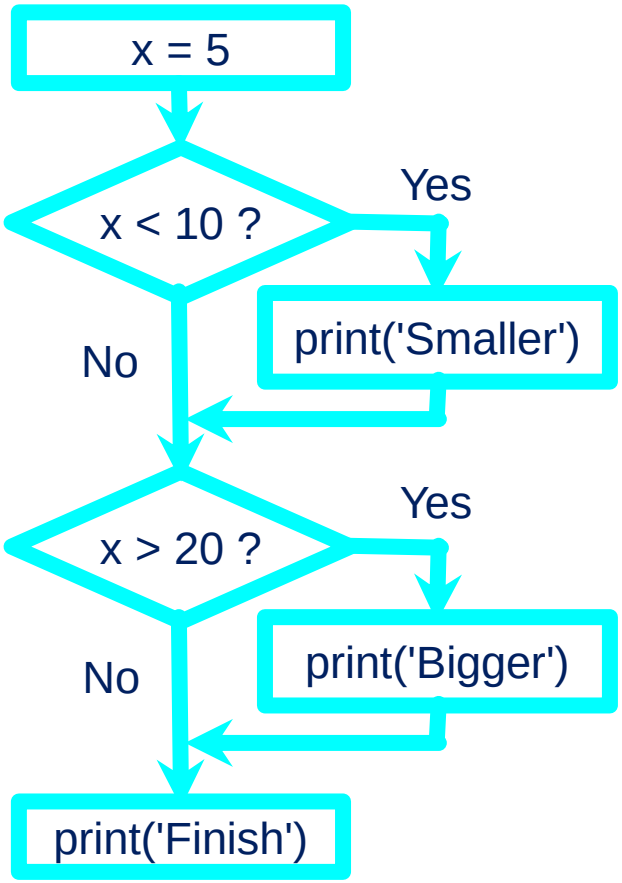
```
x = 2
print(x)
x = x + 2
print(x)
```

Output:

2
4

Una instrucción detrás de otra

2. Condicional o de Selección



TEMA 3

Program:

```
x = 5
if x < 10:
    print('Smaller')
if x > 20:
    print('Bigger')
print('Finish')
```

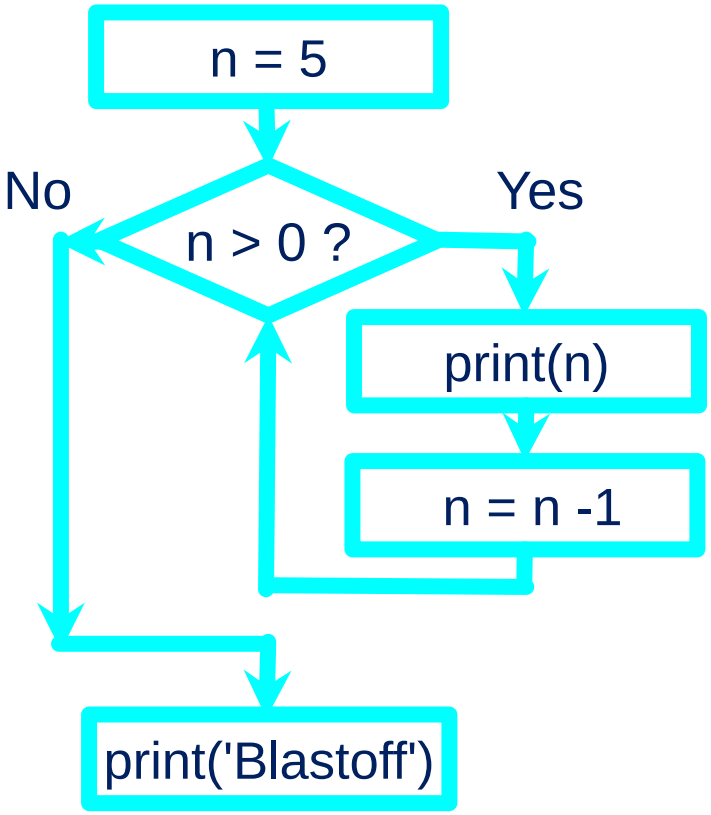
Output:

Smaller

Finish

2. Repetida

TEMA 4



Program:

```
n = 5
while n > 0 :
    print(n)
    n = n - 1
print('Blastoff!')
```

Output:

5
4
3
2
1
Blastoff!

TEMA 3

Índice

3. Instrucción condicional simple

```
if condicion1:
      accion1
      accion2
```

condición	true	false
	accion1 accion2 …	-

```
if condicion1:
      accion1
      accion2
else:
      otraAccion1
      otraAccion2
```

condición	true	false
	accion1 accion2 …	otraAccion1 otraAccion2 …

3. Instrucción condicional simple. Ejemplo

Obtener en la variable `max` el mayor de los dos números en `x` e `y`:

```
if x > y:
    max = x
else:
    max = y
```

```
if x >= y:
    max = x
else:
    max = y
```

```
max = y
if x > y:
    max = x
```

```
x = int(input('Introduce el primer numero: '))
y = int(input('Introduce el segundo numero: '))
if x > y:
    max = x
else:
    max = y
print('El mayor es... ', max)
```

```
>>> %Run max.py
  Introduce el primer numero: -9
  Introduce el segundo numero: 5
  El mayor es 5

>>>
```

3. Instrucción condicional simple. Otro ejemplo

```
num = int(input("Enter a number: "))
mod = num % 2
if mod > 0:
    print("You picked an odd number.")
else:
    print("You picked an even number.")
```

```
>>> %Run even-odd.py

  Enter a number: 45
  You picked an odd number.

>>> %Run even-odd.py

  Enter a number: 240
  You picked an even number.
```

3. Instrucción condicional simple. Otro ejemplo

```
year = int(input('Escribe un año: '))
esBiesto = (year % 4 == 0)
esBiesto = esBiesto and ((year % 100) != 0)
esBiesto = esBiesto or ((year % 400) == 0)
print(esBiesto)
```

```
year = int(input('Escribe un año: '))
if ((year % 4 == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(True)
else:
    print(False)
```

Un año es bisiesto si es divisible por 4 y no divisible por 100, excepto si es también divisible por 400, en cuyo caso es bisiesto.

TEMA 3

Índice

4. Instrucciones condicionales anidadas

```
if condicion1:
    if condicion2:
        accion1
        …
    else:
       accion2
       …
```

condicion1	condicion2	
	true	false
true	accion1	accion2
false	--	--

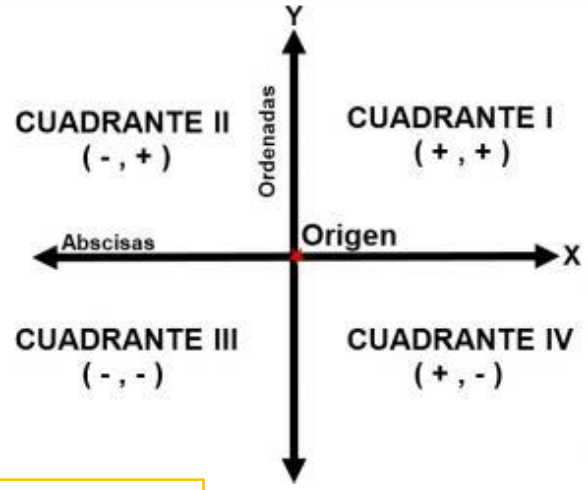
```
if condicion1:
    if condicion2:
        accion1
        …
    else:
        accion2
        …
else:
    accion3
    …
```

condicion1	condicion2	
	true	false
true	accion1	accion2
false	accion3	accion3

Cualquier instrucción puede ser a su vez una instrucción condicional: diferentes niveles de anidamiento

4. Instrucción condicional anidada. Ejemplo

Leer las coordenadas de un punto y comprobar si dicho punto esta en el centro, sobre el eje de abscisas, sobre el de ordenadas o en algún cuadrante.




```
x = int(input('Introduce la abscisa: '))
y = int(input('Introduce la ordenada: '))
if x == 0:
    if y == 0:
        print('En el centro de coordenadas')
    else:
        print('En el eje de ordenadas')
else:
    if y == 0:
        print('En el eje de abscisas')
    else:
        print('En un cuadrante')
```

TEMA 3

Índice

5. Instrucción condicional múltiple (if—elif).

```
if condicion1:
    accion1
else:
    if condicion2:
        accion2
    else:
        if condicion3:
            accion3
        else:
            accion4
```

```
if condicion1:
    accion1
elif condicion2:
    accion2
elif condicion3:
    accion3
else:
    accion4
```

condicion1	condicion2	condicion3	instrucción
true	--	-	accion1
false	true	-	accion2
false	false	true	accion3
false	false	false	accion4

El número de caminos puede ser cualquiera

5. Instrucción condicional múltiple (if—elif). Ejemplo

Escribe un programa que, dada una nota escribe por pantalla la calificación en letra: si la nota es inferior a 5 suspenso, entre 5 y 7 aprobado, entre 7 y 9 notable, entre 9 y 10 sobresaliente y 10 matrícula de honor.

```
nota = float(input('Introduce la nota (valor numérico): '))
if nota < 0.0 or nota > 10.0:
    notaLetra = 'Error'
elif nota < 5.0:
    notaLetra = 'Suspenso'
elif nota < 7.0:
    notaLetra = 'Aprobado'
elif nota < 9.0:
    notaLetra = 'Notable'
elif nota < 10.0:
    notaLetra = 'Sobresaliente'
else:
    notaLetra = 'Matrícula de Honor'
print('la calificación es... ', notaLetra)
```

5. Instrucción condicional múltiple (if—elif). Ejemplo

```
year = int(input('Escribe un año: '))
esBiesto = (year % 4 == 0)
esBiesto = esBiesto and ((year % 100) != 0)
esBiesto = esBiesto or ((year % 400) == 0)
print(esBiesto)
```

```
year = int(input('Escribe un año: '))
if ((year % 4 == 0) and ((year % 100) != 0)) or ((year % 400) == 0):
    print(True)
else:
    print(False)
```

```
year = int(input('Escribe un año: '))
if ((year % 400) == 0):
    print(True)
elif ((year % 100) == 0):
    print(False)
elif (year % 4 == 0):
    print(True)
else: print(False)
```

Un año es bisiesto si es divisible por 4 y no divisible por 100, excepto si es también divisible por 400, en cuyo caso es bisiesto.

ejercicios reader 6 – 16

(el 17 es más difícil ;-)

TEMA 3

Índice

6. Excepciones

- El flujo también se puede alterar con instrucciones de emisión y captura de excepciones

- Excepciones permiten señalar, detectar y tratar errores que se producen al ejecutar un programa.

6. Tratamiento de excepciones

- Una **Excepción** es una situación anómala ocurrida durante la ejecución de un programa.

- Ejemplos:
  - División por cero
  - Acceso a una posición de una lista inexistente
  - Abrir un fichero para leer o escribir sin permisos para ello
  - Usar como argumento de una función un valor incorrecto

- Python tiene la estructura de control **`try-except`** para la detección y tratamiento de excepciones.

6. Tratamiento de excepciones: `try-except`

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

```
1  try:
2      x = int(input("Please enter a number: "))
3      print("Dividing 50 by", x,"will give you :", 50/x)
4  except:
5      print("Something went wrong, closing...")
```

Shell

```
>>> %Run ejemplo1-try.py
  Please enter a number: 78
  Dividing 50 by 78 will give you : 0.6410256410256411
>>> %Run ejemplo1-try.py
  Please enter a number: Hola!
  Something went wrong, closing...
>>>
```

6. Tratamiento de excepciones

- Las **excepciones** tienen un **tipo asociado** y podemos distinguir el tipo de excepción para actuar de diferente forma en función del tipo de error detectado

- Ejemplos:
  - Una división por cero es un error de tipo **`ZeroDivisionError`**.
  - El acceso a un index inexistente es un error de tipo **`IndexError`**.
  - Abrir un fichero para leer que no existe es un error de tipo **`FileNotFoundError`**.
  - el intento de calcular la raíz cuadrada de un valor negativo es un error de tipo **`ValueError`**.

- El intérprete de Python resulta útil para recordar el tipo del error que deseamos tratar.

6. Tratamiento de excepciones: try-except

```
1  print('Solución de la ecuación lineal ax + b = 0')
2  a = float(input('Introduce el valor de a: '))
3  b = float(input('Introduce el valor de b: '))
4  try:
5      x = - b / a #Si a=0 excepción de división por 0
6      print('Solución: ', x)
7  except:
8      if b != 0:
9          print('No tiene solución.')
10     else:
11         print('Tiene infinitas soluciones.')
```

Shell

```
>>> %Run ecuacion-lineal.py

 Solución de la ecuación lineal ax + b = 0
 Introduce el valor de a: 2
 Introduce el valor de b: 3
 Solución:  -1.5

>>> %Run ecuacion-lineal.py

 Solución de la ecuación lineal ax + b = 0
 Introduce el valor de a: 0
 Introduce el valor de b: 5
 No tiene solución.

>>> %Run ecuacion-lineal.py

 Solución de la ecuación lineal ax + b = 0
 Introduce el valor de a: 0
 Introduce el valor de b: 0
 Tiene infinitas soluciones.
```

Tratamiento de excepciones: try-except

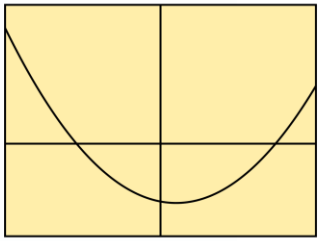
```python
import math

a = float(input('Valor de a: '))
b = float(input('Valor de b: '))
c = float(input('Valor de c: '))

try: #Si a=0 excep. de división por 0. Raíz de número negativo excep. valor erróneo.
    x1 = (-b + math.sqrt(b**2 - 4*a*c)) / (2 * a)
    x2 = (-b - math.sqrt(b**2 - 4*a*c)) / (2 * a)
    if x1 == x2:
        print('Solución doble x =', x1)
    else:
        print('Soluciones de la ecuacion: x1=%4.3f y x2=%4.3f ' % (x1, x2))
except ZeroDivisionError:
    if b != 0:
        print('Solucion de la ecuacion: x=%4.3f ' % -c/b)
    elif (c == 0):
        print('La ecuacion tiene infinitas soluciones.')
    else:
        print('La ecuacion no tiene solucion.')
except ValueError:
    print('La ecuacion no tiene soluciones reales.')

```

Shell

```
>>> %Run ecuacion-cuadratica.py

  Valor de a: 1
  Valor de b: 6
  Valor de c: 4
  Soluciones de la ecuacion: x1=-0.764 y x2=-5.236

>>> %Run ecuacion-cuadratica.py

  Valor de a: 1
  Valor de b: 4
  Valor de c: 4
  Solución doble x = -2.0

>>> %Run ecuacion-cuadratica.py

  Valor de a: 1
  Valor de b: 2
  Valor de c: 3
  La ecuacion no tiene soluciones reales.
```

Ecuación cuadrática

$$ax^2 + bx + c = 0$$



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

Otras: Raise, as, finally

Si no puede convertir a float lanza **ValueError**

```
try:
    v1 = float( input( "Introduce el valor 1: " ) )
    v2 = float( input( "Introduce el valor 2: " ) )

    op = input( "Introduce operacion + - * / :" )
    if   op == "+":
        print( "Resultado: ", v1 + v2 )
    elif op == "-":
        print( "Resultado: ", v1 - v2 )
    elif op == "*":
        print( "Resultado: ", v1 * v2 )
    elif op == "/":
        print( "Resultado: ", v1 / v2 )
    else:
        raise Exception( "Operador incorrecto" )

except ValueError:
    print( "Not a float" )
except ZeroDivisionError:
    print("Division por cero")
except Exception as e:
    print( e )
finally:
    print("Esto siempre se ejecuta, se capture excepción o no")
```

Con **raise** podemos lanzar excepciones personalizadas

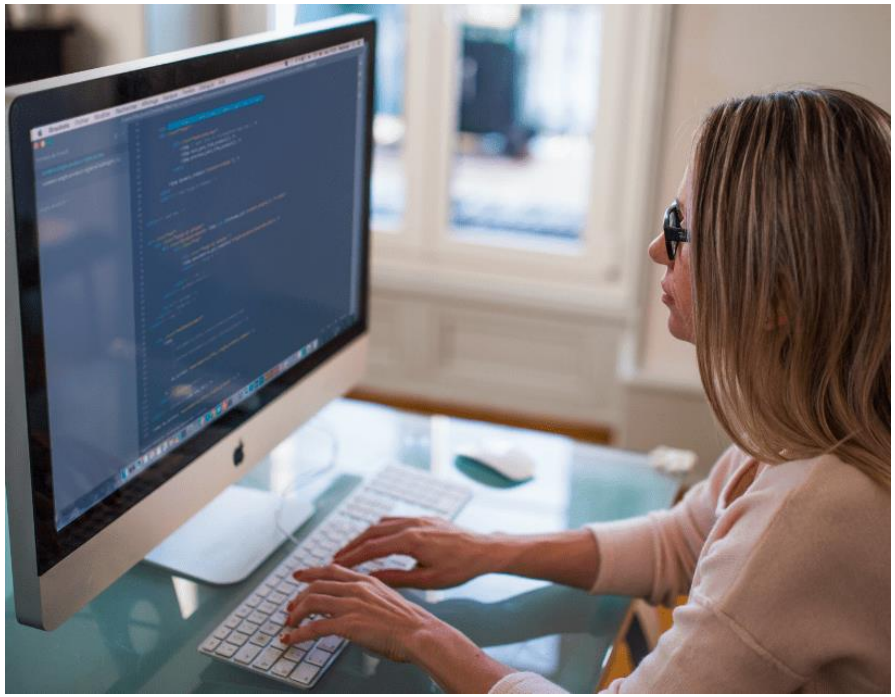
Exception es el tipo más genérico, como último except captura cualquiera que no aparezca en except previos

Con **as** podemos recuperar el mensaje de la excepción, metiéndola en la variable **e**

Las sentencias dentro de **finally** se ejecutan siempre al terminar, se haya capturado excepción o no.

ejercicio 18

¡Solo hay una forma de aprender a programar!



practicando, practicando, practicando, practicando