Parte I: Programación en un lenguaje orientado a objetos

1. Introducción a los lenguajes de programación

2. Datos y expresiones

- Números. Operaciones y expresiones. Variables. Booleanos. Strings. Uso de funciones matemáticas. Variables y paso de parámetros. Listas y tuplas.

3. Clases

4. Estructuras algorítmicas

5. Estructuras de Datos

6. Tratamiento de errores

7. Entrada/salida

8. Herencia y polimorfismo

 © Michael González Harbour, 28/feb/24

2.1. Números

Python dispone de tres tipos de números con los que podemos trabajar

- `int`: números enteros
  - precisión ilimitada
  - aritmética *exacta*
- `float`: números reales
  - número real de unos 15 dígitos de precisión y rango aprox. ±1.7E+308
  - ocupa 64 bits (8bytes)
  - aritmética *aproximada* (errores de redondeo)
  - admite valores especiales como `+∞`, `-∞` o `nan` (*not a number*)
- `complex`: números complejos
  - formados por dos números reales que actúan como parte real y parte imaginaria

© Michael González Harbour, 28/feb/24

Literales de los números

Permiten expresar valores fijos

tipo	descripción	ejemplos

`int`	el número entero en decimal
	el número en octal, con `0o`
	el número en hexadecimal, con `0x`
	el número en binario, con `0b`
		`12 -37 1_000_000`
		`0o37`
		`0x2A`
		`0b100101`

`float`	el número con parte fraccionaria y/o notación exponencial con `e` o `E`	`0.0 13.56 -12.0` `6.023E23 1.0e-6`
`complex`	se añade la letra `j` para indicar la parte imaginaria	`1 + 2j` `0.1 + 2.5j`

2.2. Operaciones y expresiones

Las *expresiones* permiten transformar datos para obtener un resultado

Se construyen con *operadores* y *operandos*

Operandos (datos):

- constantes literales
- datos de los tipos predefinidos (atributos, variables, o argumentos de tipos predefinidos)
- funciones, cuando retornan un valor de un tipo predefinido

© Michael González Harbour, 28/feb/24

Operaciones aritméticas

Principales operadores aritméticos: operan con números

Operación	Resultado
`x + y`	suma de x e y
`x - y`	diferencia x menos y
`x * y`	producto de x por y
`x / y`	cociente de x entre y; si los operandos son enteros, el resultado es *real*
`x // y`	cociente entero de x entre y; si los operandos son enteros, el resultado se trunca y es *entero*
`x % y`	módulo: resto de la división entera de x entre y
`-x`	cambio de signo de x
`abs(x)`	valor absoluto de x
`round(x)`	redondeo al entero más próximo
`x ** y`	x elevado a y

 © Michael González Harbour, 28/feb/24

Precedencia

Cuando aparecen varios operadores en una expresión se evalúan en el orden marcado por su precedencia

- es el orden habitual de la notación matemática
- para la misma precedencia se usa orden de izquierda a derecha
- Ejemplo

`4+5*2**2`	$4 + 5 \cdot 2^2 = 24$

La precedencia se puede alterar con paréntesis (no `[ ]` ni `{}`)

`(4+5)*2**2`	$(4 + 5) \cdot 2^2 = 36$

Tabla de precedencia

De mayor a menor

Operador	
**	
+, -, ~	unario
*, /, //, %	
+, -	binario
<<, >>	
&	
^	
|	
==, !=, >, >=, <, <=, **is**, **is not**, **in**, **not in**	
not	
and	
or	

Promoción de tipos

Cuando se mezclan números de diferente tipo, los tipos menos precisos se convierten automáticamente al tipo más preciso

El orden de menos preciso a más preciso es:

- `int => float => complex`

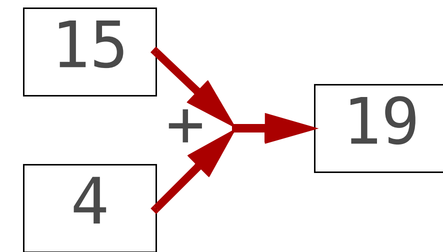
Ejemplo:

- `5+4.5` da como resultado `9.5`
- `6+(7+8j)` da como resultado `(13+8j)`

2.3. Variables

Los datos son casillas de memoria que contienen un *valor* de un *tipo* y se almacenan en la memoria en lugares llamados *objetos*

Habitualmente se crean con un literal o como resultado de una operación



15

15+4

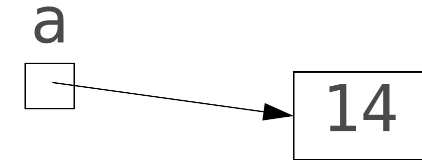
Casi todos los objetos son *inmutables* (su valor no puede cambiar)

Es habitual referirse a un dato mediante una *variable*:

- *Etiqueta* que se refiere a un dato u objeto guardado en la memoria
- Su tipo es el del dato al que se refiere
- Internamente es una *referencia* al dato
  - información que indica en qué lugar de la memoria está el objeto

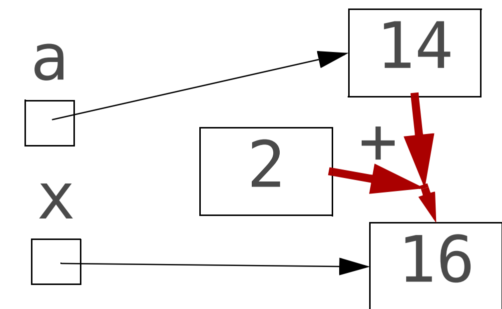
Variables y asignación

Para asignar un dato a una variable usamos el símbolo de asignación =



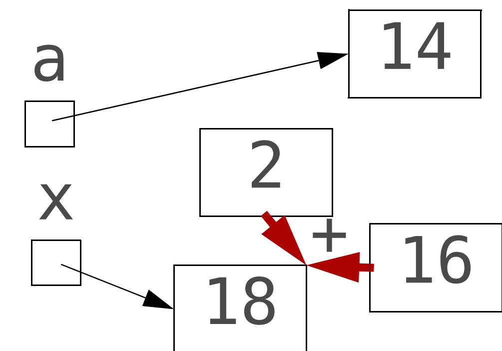
`a = 14`	asigna el dato 14 a la variable a si a no existe, se crea

donde la variable se pone siempre a la izquierda



`x = a+2`	asigna el resultado de la suma (16) a la variable x

¡Cuidado! = no es una igualdad matemática



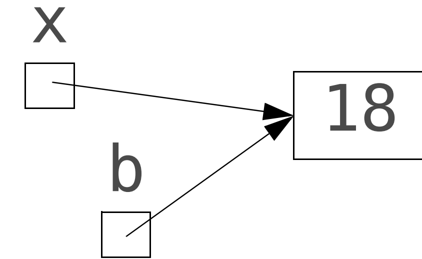
`x = x+2`	asigna el dato 18 a la variable x

El código es secuencial: una variable (o cualquier nombre) no se puede usar antes de crearla

© Michael González Harbour, 28/feb/24

Asignación (cont.)

La asignación puede crear variables, pero no crea objetos



`b = x`	asigna a la variable b el *mismo* dato (18) asignado a x

Conceptualmente:

`objeto` → `dato`

`variable` → `etiqueta`

© Michael González Harbour, 28/feb/24

Instrucciones de asignación

Hay instrucciones que combinan asignación y una operación aritmética

La asignación se hace después de calcular la expresión de la derecha

Instrucción	Resultado
`x = expr`	asigna a `x` la expresión de la derecha
`x += expr`	asigna a `x` la suma de `x` más la expresión de la derecha equivale a `x = x+expr`
`x -= expr`	asigna a `x` la resta de `x` menos la expresión de la derecha
`x *= expr`	asigna a `x` el producto de `x` por la expresión de la derecha
`x /= expr`	asigna a `x` el cociente de `x` entre la expresión de la derecha
...	hay operadores combinados con todos los operadores aritmé-ticos

Ejemplo que incrementa el valor asignado a `x`

```
x = x+1        o        x+=1
```

Tipos de variables por su ámbito

Variables *locales*

- definidas directamente en el interior de una función
- tienen una vida corta, solo mientras se ejecuta la función
  - aunque si la función es el programa principal (`main`) entonces su vida es la del programa, y por tanto larga
- se destruyen al acabar la función
- se usan solo para cálculos provisionales

Argumentos o parámetros

- definidos en los paréntesis del encabezamiento de la función
- se les da valor al invocar la función
- representan datos que la función obtiene del exterior
- su vida es corta, como las variables locales

© Michael González Harbour, 28/feb/24

Tipos de variables por su ámbito (cont.)

Variables *globales*

- definidas directamente en un módulo o fichero
- tiene una vida larga, igual a la del módulo
- su ámbito amplio puede provocar modificaciones *no intencionadas* por lo que están desaconsejadas; *debemos evitarlas, excepto si son constantes*

Entonces, ¿cómo creamos variables de vida larga?

Atributos

- definidas en el interior de las clases (que veremos más adelante)
- tienen una vida larga; están disponibles siempre que se necesiten
- su ámbito está restringido a la clase, lo que evita el peligro

© Michael González Harbour, 28/feb/24

Tipos de variables por su ámbito (cont.)

variable global
vida larga
¡peligrosa si no es constante!

argumento
vida corta
existe solo mientras se ejecuta **func**
su valor viene de fuera de **func**

variable local
vida corta
existe solo mientras se ejecuta **func**
su valor viene del interior de **func**

```python
# -*- coding: utf-8 -*-
"""
Programa que ilustra ámbitos de variables

@author: Michael González
@date  : 17/ene/2024
"""

g: float = 0.0

def func(x: float):
    """
    Muestra en pantalla x+1
    """

    y: float = 1.0
    print(x+y)

def main():
    """
    Programa principal
    """
    func(12.0)
```

Asociación de parámetros al invocar una función

Los parámetros se definen en la cabecera de la función

La asociación de valores a los parámetros se hace al invocar la función, por su orden

```
mi_funcion(12, 1.5, 7.8)
```

por orden

```
def mi_funcion(num: int, masa: float, vel: float):
```

```
mi_funcion(vel=7.8, num=12, masa=1.5)
```

por nombre

O, alternativamente, dando sus nombres: el orden no importa

© Michael González Harbour, 28/feb/24

Asociación de parámetros al invocar una función (cont.)

Los valores asociados a los parámetros al invocar la función pueden ser literales, variables o expresiones

```
mi_funcion(12, variable_1, a+10*b)
```

Ejemplo de un programa con datos

Cálculo de la media de tres notas

- las notas (variables enteras) deben tener vida larga
- como aún no hemos visto las clases serán variables del `main()`

El módulo tendrá:

- función `media_entera`:()
  - recibe las tres notas como parámetros y retorna la media *entera*
- función `media_real()`:
  - recibe las tres notas como parámetros y retorna la media *real*
- una función `main()` con tres variables enteras (las tres notas)

Podríamos hacer todos los cálculos en el `main`, sin otras funciones

- pero es conveniente aplicar el principio de "*divide y vencerás*"
- dividiendo la funcionalidad en varias partes

Diagrama de clases

Podemos interpretar el módulo como una clase

	notas
nombre del módulo	**notas**
atributos	
funciones	media_entera(nota_1:int, nota_2:int, nota_3:int): int media_real(nota_1:int, nota_2:int, nota_3:int): float main()

Observar los encabezamientos de funciones:

- formato de los argumentos: `nombre: tipo`
- valor retornado (en su caso): `encabezamiento: tipo`

Ejemplo (cont)

```python
# -*- coding: utf-8 -*-
"""
Opera con las notas de tres alumnos

@author: Michael
@date  : ene-2024
"""
```


© Michael González Harbour, 28/feb/24

Ejemplo (cont)

```python
def media_entera(nota_1: int, nota_2: int,
                 nota_3: int) -> int:
    """
    Calcula la media de tres notas como numero entero,
    truncando el resultado

    Args:
        nota_1: nota del primer examen
        nota_2: nota del segundo examen
        nota_3: nota del tercer examen
    Returns:
        la media entera de las tres notas

    """

    return (nota_1+nota_2+nota_3)//3
```

Ejemplo (cont)

```
def media_real(nota_1: int, nota_2: int,
               nota_3: int) -> float:
    """
    Calcula la media de tres notas como numero real

    Args:
        nota_1:  nota del primer examen
        nota_2:  nota del segundo examen
        nota_3:  nota del tercer examen
    Returns:
        la media real de las tres notas
    """

    return (nota_1+nota_2+nota_3)/3
```

Ejemplo (cont)

```python
def main():
    """
    Programa principal

    Contiene tres notas de un alumno y muestra su media
    de dos formas
    """

    # Declaramos las notas y les damos valor
    nota_1: int = 7
    nota_2: int = 8
    nota_3: int = 8

    # Mostramos resultados
    media: int = media_entera(nota_1, nota_2, nota_3)
    print("Nota media entera =", media)

    # Esta vez sin usar una variable intermedia
    print("Nota media real   =",
          media_real(nota_1, nota_2, nota_3))
```

Orden de ejecución

Es importante darse cuenta que el orden de ejecución lo definen las instrucciones del programa principal, `main()`

- A pesar de que las funciones `media_entera()` y `media_real()` están escritas antes que el `main()`, sus instrucciones solo se ejecutan cuando el `main()` las invoca

© Michael González Harbour, 28/feb/24

Documentación de funciones

Los *docstrings* que documentan las funciones tienen estos requisitos:

- Mostrar una descripción breve de lo que hace la función
  - No cómo está hecha. Esto se hace en los comentarios internos
- En su caso, mostrar una descripción de cada argumento con su nombre y lo que significa, sin olvidar describir sus unidades si las tiene
- En su caso, mostrar una descripción de lo que retorna la función y de sus unidades si las tiene

En el ejemplo se muestra el estilo de documentación de Google

- Hay otros estilos frecuentes
- Se puede configurar en *Spyder*, para facilitar la documentación

© Michael González Harbour, 28/feb/24

Otros comentarios sobre el ejemplo

- argumentos de un método: datos del exterior, que el método necesita
- instrucción `return`, para expresar el valor de retorno de un método (su respuesta)
- operador de asignación: "`=`"
- operador de suma aritmética: "`+`"
- `print()` con varios parámetros, separados por comas
- uso de paréntesis
- divisiones reales y enteras
- comentarios de documentación e internos
- anotaciones de tipo
  - en las variables: `variable: tipo`
  - en el valor de retorno de la función: `encabezamiento-> tipo:`

Comentarios internos

- ayudan a entender el código
- formato: texto entre # y el final de la línea
- se suelen poner encima de la instrucción o instrucciones a las que afecta

```
# Esto es un comentario
instrucciones a las que afecta
```


© Michael González Harbour, 28/feb/24

Constantes

Algunos lenguajes disponen de constantes, similares a las variables pero a las que no se puede cambiar el dato asignado

En Python no hay constantes, pero para indicar al programador que se *desea* que a una variable no pueda asignársele un valor nuevo se recurre a una convención

- se pone su nombre en mayúsculas
- Ejemplo:

```
CTE_DE_PLANK: float = 6.626070150E-34 # J*s
```

Se pueden definir como variables globales, ya que (se supone que) nadie les cambiará su valor

Es muy importante usar constantes para mejorar la legibilidad, para cualquier valor literal que preveamos que podría cambiar, o para cualquier valor literal que se repita

Constantes a partir de Python 3.8

Además, a partir de Python 3.8 es posible indicar de una forma más explícita que deseamos que una variable tenga un valor constante

Para ello, se pone una anotación de tipo con el formato `Final[tipo]`

- Que indica que la variable toma su valor final

Para poder usar la anotación `Final`, es preciso poner al principio del módulo una instrucción de tipo **`import`**, como en el ejemplo:

```
from typing import Final

GRAVITACION_UNIV: Final[float] = 6.674e-11 #N*m**2*kg**-2
```

Intentaremos usar esta anotación de tipos siempre que se pueda

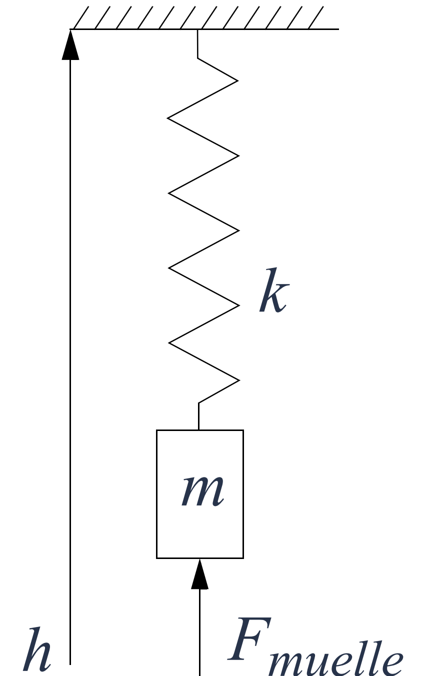
© Michael González Harbour, 28/feb/24

Ejemplo

Cálculo de la trayectoria de una masa suspendida de un muelle

Usaremos la ley de Hooke que nos da la fuerza del muelle en función de la altura (h)

$$F_{muelle} = -k \cdot h$$



© Michael González Harbour, 28/feb/24

Diseño del módulo:

Diagrama de clases

- constantes para la constante del muelle y la gravedad
- una función que devuelve la fuerza del muelle, dada la altura
- una función que calcula la *nueva* altura y velocidad, dados los valores *actuales* de altura y velocidad, la masa y un intervalo de tiempo
  - retorna dos valores (una tupla)
- una función `main`

muelle
CTE_ELASTICA: float G: float
fuerza_muelle(alt: float): float avanza_tiempo(alt: float, vel: float masa: float, tiempo: float): (float, float) main()

Una *tupla* es una secuencia inmutable de elementos. Ejemplo

```
mi_tupla = (1, 2, 3, "juan")
```

Ejemplo (cont.)

```
# -*- coding: utf-8 -*-
"""
Simula el movimiento oscilatorio de una masa que
pende de un muelle, sin considerar rozamientos

Se usan unidades del sistema internacional:
    kg para la masa
    m para la altura

El cero de altura se considera en el punto en el que el muelle
está en reposo. El sentido de la fuerza y altura es positiva
hacia arriba. Los datos principales son:
    alt   : altura actual
    vel   : velocidad actual de la masa, en m/s
            inicialmente es cero
    masa  : masa del objeto
    tiempo: tiempo transcurrido

@author: Michael
@date  : Feb-2024
"""
```


© Michael González Harbour, 28/feb/24

Ejemplo (cont.)

```
from typing import Final

# Constantes:
#    CTE_ELASTICA: Constante elástica del muelle, en N/m
#    G: Gravedad terrestre, en m/s**2

CTE_ELASTICA: Final[float] = 0.12

G: Final[float] = 9.8
```

Ejemplo (cont.)

```
def fuerza_muelle(alt: float) -> float:
    """
    Calcula la fuerza que ejerce el muelle
    dada la altura de su extremo

    Args:
        alt: la altura del extremo del muelle, en m
    Returns:
        la fuerza que ejerce el muelle, en N
    """

    # retornamos la fuerza calculada con la ley de Hooke
    return -CTE_ELASTICA*alt
```


© Michael González Harbour, 28/feb/24

Ejemplo (cont.)

```python
def avanza_tiempo(alt: float, vel: float, masa: float,
                  tiempo: float) -> tuple[float, float]:
    """
    Calcula la nueva altura y velocidad de la masa,
    transcurrido el tiempo especificado, que debe ser pequeño

    Args:
        alt: la altura actual del extremo del muelle, en m
        vel: la velocidad actual del extremo del muelle, m/s
        masa: la masa del cuerpo que pende del muelle, en kg
        tiempo: el tiempo transcurrido, en s
    Returns:
        la nueva altura y velocidad de la masa (m y m/s)
    """
    # Calculamos la fuerza aplicada sobre la masa, en N
    fuerza: float = fuerza_muelle(alt)-masa*G
    # obtenemos la aceleración por la ley de Newton
    aceleracion: float = fuerza/masa
    # ecuaciones del movimiento uniformemente acelerado
    return (alt+vel*tiempo+(aceleracion*tiempo**2)/2,
            vel+aceleracion*tiempo)
```

Ejemplo (cont.)

```
def main():
    """
    Simula el movimiento de una masa suspendida de un
    muelle durante un tiempo, y pone en pantalla
    altura y velocidad. Unidades del S.I
    """

    masa: float = 0.25
    incremento: float = 0.01
    alt: float = 0.2
    vel: float = 0
    # Avanzar la simulación
    (alt, vel) = avanza_tiempo(alt, vel, masa, incremento)

    # mostrar resultados con un f-string

    print(f"Alt= {alt} m. Vel= {vel} m/s")
```

observar la 'f'

Se reemplaza por el valor de alt

Se reemplaza por el valor de vel

A observar en este ejemplo:

- creación de constantes, expresadas directamente en el módulo
- creación de variables locales
- uso de *f-strings* (o strings con formato) para mostrar varios datos
  - delante del string se pone una `f` o `F`
  - las variables a mostrar en mitad del texto se encierran entre `{}`
- uso de una tupla con dos valores
- comentarios de documentación con:
  - explicación general
  - explicación de los parámetros con sus unidades
  - explicación del valor retornado con sus unidades

© Michael González Harbour, 28/feb/24

2.4 Booleanos

El tipo `bool` permite representar los valores lógicos verdad (`True`) o falso (`False`)

- también llamamos a este tipo de datos *booleano*

Es extremadamente importante, pues casi todos los programas deben tomar decisiones, que habitualmente se basan en valores lógicos

Una forma habitual de obtener valores booleanos es mediante los operadores de comparación o relacionales

- el resultado de la comparación es un booleano

Operadores relacionales

Operación	Resultado (`True` si se cumple la condición, `False` en caso contrario)
`x == y`	x es igual a y
`x != y`	x es distinto de y
`x > y`	x es mayor que y (estrictamente mayor)
`x >= y`	x es mayor o igual que y
`x < y`	x es menor que y (estrictamente menor)
`x <= y`	x es menor o igual que y
`x is y`	x e y están asignadas al mismo objeto
`x is not y`	x e y no están asignadas al mismo objeto

Operadores lógicos

Podemos trabajar con los operadores lógicos para escribir expresiones lógicas a partir de datos booleanos

Operación	Resultado (`True` si se cumple la condición, `False` en caso contrario)
`x or y`	o lógico: si x es False, entonces y, si no, x
`x and y`	y lógico: si x es False, entonces x, si no, y
`not x`	negación: si x es False, entonces True, si no, False

- por eficiencia, las operaciones **`or`** y **`and`** son condicionales:
  - **`or`**: el elemento derecho solo se evalúa si el izquierdo es `False`
  - **`and`**: el elemento derecho solo se evalúa si el izquierdo es `True`
- **`not`** tiene menor precedencia que los operadores no booleanos
  - `not a == b` se interpreta como `not (a == b)`
  - `a == not b` es un error de sintaxis

Ejemplo con booleanos

Escribiremos un programa para determinar si un año entre el 2000 y 2099 cuyo valor introducimos por teclado es bisiesto o no

- el año es bisiesto si es múltiplo de `4`

Para determinar si un número es múltiplo de otro usaremos el operador módulo (`%`)

- si el resto de `a/b` es cero, `a` es múltiplo de `b`

```
a%b == 0
```


© Michael González Harbour, 28/feb/24

Ejemplo

```
# -*- coding: utf-8 -*-
"""
Programa que determina si un año es bisiesto

Contiene una función para determinar si un año es bisiesto
y un main que lee el año del teclado y muestra el resultado

@author: Michael
@date  : Feb 2024
"""
```


© Michael González Harbour, 28/feb/24

Ejemplo (cont.)

```python
def es_bisiesto(year: int) -> bool:
    """
    Determina si un año entre el 2000 y el 2099 es bisiesto

    Para otros años hay que usar reglas más complejas

    Args:
        year: el año

    Returns:
        un booleano que indica si el año es bisiesto o no
    """

    return year%4 == 0
```

Ejemplo (cont.)

```python
def main():
    """
    Programa que lee un año de teclado y
    muestra en pantalla si es bisiesto o no
    """

    # Lee el año del teclado
    year: int = int(input("Introduce el año:"))

    # Determina si es bisiesto o no
    bisiesto: bool = es_bisiesto(year)

    # Muestra el resultado en la pantalla con f-strings
    print(f"El año {year} es bisiesto: {bisiesto}")
```


© Michael González Harbour, 28/feb/24

A observar en el ejemplo

- uso de expresiones relacionales
- uso de variables booleanas
- lectura del teclado
  - con `input()` leemos un string (texto)
  - con `int(texto)` convertimos el texto a entero
  - se podría hacer con `float(texto)` para convertir a real

© Michael González Harbour, 28/feb/24

2.5 Strings (Textos)

El tipo de datos str permite almacenar secuencias de cero o más caracteres

Los literales de string se forman poniendo texto entre comillas

Strings unilínea	`'Un texto'`
	`"otro texto"`
Strings multilínea	`'''otro más'''`
	`"""y otro más"""`

Operador de concatenación: crea un nuevo string a partir de otros dos,

```
"Ana"+" está aquí"  # resultado: "Ana está aquí"
x = 8.5
"Valor :"+x  # incorrecto; usar "Valor :"+str(x)
```

Los strings son *inmutables*

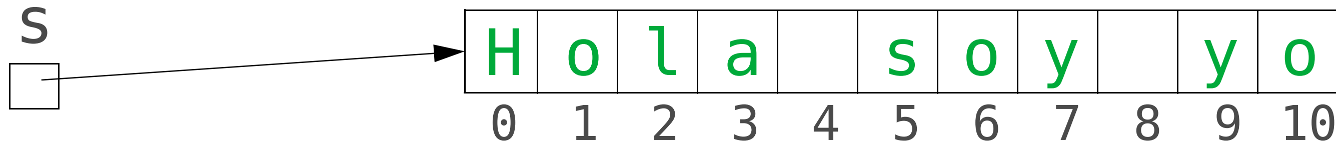
© Michael González Harbour, 28/feb/24

Ejemplo de programa con variables de texto

```python
# -*- coding: utf-8 -*-
"""
Programa para trabajar con el nombre de una persona y el de su
padre

@author: Michael
@date  : Feb 2024
"""

def main():
    """
    Programa que pregunta dos nombres por teclado y
    los muestra en pantalla concatenados
    """

    nombre: str = input("Introduce tu nombre: ")
    padre: str = input("El nombre de tu padre: ")

    print(f"El padre de {nombre} es {padre}")
```

Manipulación de strings

Los caracteres se numeran empezando en cero



Obtener el carácter número `i` de un string: `s[i]`

```
s[3] # "a"
```

Obtener un fragmento de un string: `s[i:j] # i incl., j excl.`

```
s[0:4] # "Hola"
```

Saber si un string contiene a otro:

```
'yo' in s # True
```

Obtener la longitud de un string:

```
len(s) # 11
```


© Michael González Harbour, 28/feb/24

Manipulación de strings (cont.)

Los strings tienen métodos, que son funciones que se aplican directamente al objeto con la notación

```
texto.método(parámetros)
```

Algunas operaciones usuales, siendo `s` un string

`s.find(sub)`	Retorna el primer índice en el que se encuentra `sub` dentro de `s`. Si no se encuentra, retorna `-1`
`s.strip()`	Retorna una copia de `s` sin los espacios en blanco iniciales ni finales
`s.lower()`	Retorna una copia en minúsculas de `s`
`s.upper()`	Retorna una copia en mayúsculas de `s`
`s.startswith(sub)`	Retorna `True` si `s` comienza por `sub`

Ejemplo de manipulación de strings

```python
# -*- coding: utf-8 -*-
"""
Conversor de temperaturas entre Farenheit y Celsius

La conversión se hace con la fórmula: F=C*1.8+32

@author: Michael
@date:   Feb 2024
"""

def to_farenheit(cels: float) -> float:
    """
    Convierte Celsius a Farenheit

    Args:
        cels : Temperatura celsius
    Returns:
        El valor de cels en Farenheit
    """

    return cels*1.8+32
```

Ejemplo (cont.)

```
def to_celsius(faren: float) -> float:
    """
    Convierte Farenheit a Celsius

    Args:
        faren : Temperatura Farenheit
    Returns:
        El valor de faren en Celsius
    """

    return (faren-32)/1.8
```

Ejemplo (cont.)

```
def main():
    """
    Conversor entre temperaturas Farenheit y Celsius


    Se lee un texto que contiene la temperatura y la unidad,
    separados por un espacio en blanco. Ejemplos:
        20.0 C
        70.0 F
    A continuación se muestra el resultado en la otra unidad

    Por sencillez no hacemos detección de errores.
    Lo veremos más adelante
    """
```


© Michael González Harbour, 28/feb/24

Ejemplo (cont.)

```
# Leer la temperatura inicial
inicial: str = input("Temp. en C o F. Ejemplo:'30.0 C': ")
inicial = inicial.strip()

# Obtenemos la posición del espacio en blanco
indice_espacio: int = inicial.find(" ")

# Separamos el número y las unidades
temp: float = float(inicial[0:indice_espacio])
unidades: str = inicial[indice_espacio+1:].strip()

# Comparamos las unidades en mayúsculas
if unidades.upper() == "C":
    # Celsius
    print(f"Temperatura: {to_farenheit(temp)} F")
else:
    # Entendemos que las unidades son Farenheit
    print(f"Temperatura: {to_celsius(temp)} C")
```

A observar en este ejemplo

Rodajas de string

- `s[i:j]`: incluye los caracteres de `i` a `j-1`
- `s[i:]`: incluye los caracteres desde `i` hasta el *último*

Uso de `strip()` para eliminar espacios extra al principio o al final

Comparación de strings en mayúsculas, para evitar diferencias entre minúsculas y mayúsculas

Uso de la instrucción condicional:

```
if expresion_booleana:
    instrucciones  # si True
else:
    instruciones   # Si False
```

Conversión de tipos

Compatibilidad de tipos: Python es un lenguaje con tipificación estricta

- No se pueden mezclar datos de distinta naturaleza en las expresiones
  - por ejemplo, números con strings
- Los números son compatibles entre sí

En ocasiones nos interesa cambiar el tipo de un dato

Conversión explícita de tipo:

```
tipo(dato)
```

Ejemplo:

```
numero = int(texto)
```

Hay que usarlas con precaución

© Michael González Harbour, 28/feb/24

2.6. Uso de funciones matemáticas

El módulo `math` contiene constantes y funciones para hacer operaciones aritméticas con números reales (float)

- Existe un módulo similar llamado `cmath` para números complejos

Para usarlo debemos poner una instrucción `import` tras el *docstring* del módulo:

```
import math
```

Constantes:

`math.pi`	El número π
`math.e`	El número e
`math.inf`	+infinito
`math.nan`	*Not a number*, se usa para indicar errores

Operaciones matemáticas frecuentes

`math.sin(a), math.cos(a), math.tan(a)`	funciones trigonométricas (radianes)
`math.asin(v), math.acos(v), math.atan(v), math.atan2(y,x)`	trigonométricas inversas (radianes) de $-\pi/2$ a $\pi/2$ arco tangente de y/x entre $-\pi$ y π
`math.degrees(a), math.radians(a)`	conversiones de ángulos
`math.sinh(a), math.cosh(a), math.tanh(a)`	funciones trigonométricas hiperbólicas
`math.asinh(a), math.acosh(a), math.atanh(a)`	funciones trigonométricas hiperbólicas inversas
`math.exp(x), math.log(x), math.log10(x)`	e^x, logaritmo neperiano y logaritmo decimal
`math.sqrt(x)`	raíz cuadrada
`math.ceil(x), math.floor(x)`	redondeo por arriba y por abajo

Operaciones matemáticas (cont.)

`math.factorial(x)`	factorial de x (debe ser entero no negativo)
`math.copysign(x,y)`	retorna x con el signo de y
`math.dist((x1, y1), (x2, y2))`	retorna la distancia entre dos puntos[a][b] $\sqrt{(x2-x1)^2+(y2-y1)^2}$
`math.isinf(x)`	retorna True si x es ±infinito
`math.isnan(x)`	retorna True si x es nan
`math.isclose(a, b)`	retorna True si a y b están próximos; la tolerancia relativa por defecto es 1e-9, pero se puede cambiar con parámetros adicionales

a. Disponible a partir de Python 3.8

b. Se muestra un ejemplo con puntos de dimensión 2, pero la función permite también puntos de más dimensiones. Retorna la distancia euclidiana.

Funciones predefinidas

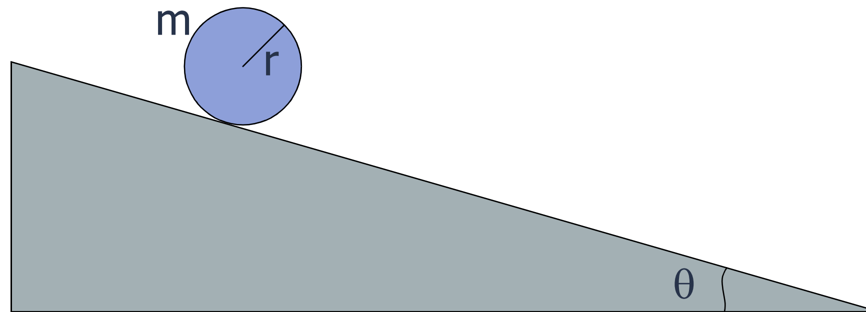
Existen funciones predefinidas (que no forman parte de `math`), pero son útiles

Ya vimos `abs()` y `round()`. Algunas más son:

`max(x,y, ...)` `min(x,y, ...)`	máximo o mínimo de dos o más valores: ambas para cualquier valor numérico
`max(iterable)` `min(iterable)`	máximo o mínimo de los elementos de una lista, tupla o secuencia iterable
`sum(iterable)`	suma de los elementos de una lista, tupla o secuencia iterable

Ejemplo con funciones trigonométricas

Cálculo de movimientos de una esfera que rueda sobre un plano inclinado



Datos del ejemplo

I	momento de inercia	v	velocidad
m	masa	t	tiempo
r	radio	x	distancia
θ	ángulo del plano inclinado	E_{tras}	energía cinética de traslación
g	gravedad	E_{rot}	energía cinética de rotación
a	aceleración	ω	velocidad angular ($v=\omega r$)

Ecuaciones del movimiento

Las ecuaciones son (`http://www.sc.ehu.es/sbweb/fisica3/`):

$$I = \frac{2}{5}mr^2 \qquad a = \frac{g\sin\theta}{\left(1 + \frac{I}{mr^2}\right)} \qquad v = at = \omega r$$

$$x = \frac{1}{2}at^2 \qquad E_{tras} = \frac{1}{2}mv^2 \qquad E_{rot} = \frac{1}{2}I\omega^2$$

© Michael González Harbour, 28/feb/24

Ejemplo: Diagrama de clases

Diseño del módulo

- la gravedad (G) es una constante
- funciones
  - cálculo de la aceleración
  - cálculo de la distancia
  - cálculo de la energía cinética de traslación
  - cálculo de la energía cinética de rotación

PlanoInclinado
G: float
aceleracion(masa:float, radio:float, angulo: float): float distancia(masa:float, radio:float angulo: float, tiempo: float): float e_cinetica_tras(masa:float, radio:float angulo: float, tiempo: float): float e_cinetica_rot(masa:float, radio:float angulo: float, tiempo: float): float

Ejemplo (cont.)

```
# -*- coding: utf-8 -*-
"""
Simula el movimiento de una esfera en un plano inclinado

Se entiende que la esfera rueda sin deslizarse
Utilizamos unidades del sistema internacional (kg, m, s, J)
Para el ángulo utilizamos grados
Ecuaciones del movimiento en :
   http://www.sc.ehu.es/sbweb/fisica/

@author: Michael
@date:   Feb 2024
"""

from typing import Final
import math

# Constantes
#   G : Gravedad en m/s**2

G: Final[float] = 9.8
```

Ejemplo (cont.)

```
def aceleracion(masa: float, radio: float,
                angulo: float) -> float:
    """
    Calcula la aceleración lineal del objeto (m/s)
    Args:
        masa: masa de la esfera, en kg
        radio: radio de la esfera, en m
        angulo: angulo del plano inclinado, en grados
    Returns:
        la aceleración de la esfera en m/s**2
    """

    momento_inercia: float = 2.0*masa*radio**2/5.0
    return (G*math.sin(math.radians(angulo)) /
        (1+momento_inercia/(masa*radio**2)))
```

Ejemplo (cont.)

```
def distancia(masa: float, radio: float, angulo: float,
              tiempo: float) -> float:
    """
    Calcula la distancia recorrida por el
    objeto en el tiempo indicado

    Args:
        masa: masa de la esfera, en kg
        radio: radio de la esfera, en m
        angulo: angulo del plano inclinado, en grados
        tiempo transcurrido, en s
    Returns:
        la distancia recorrida por la esfera
    """

    return aceleracion(masa, radio, angulo)*tiempo**2/2.0
```


© Michael González Harbour, 28/feb/24

Ejemplo (cont.)

```
def e_cinetica_tras(masa: float, radio: float, angulo: float,
                    tiempo: float) -> float:
    """
    Calcula la energía cinética de traslación
    del objeto transcurrido el tiempo indicado (Julios)

    Args:
        masa: masa de la esfera, en kg
        radio: radio de la esfera, en m
        angulo: angulo del plano inclinado, en grados
        tiempo transcurrido, en s
    Returns:
        la energía cinética de traslación de la esfera en J
    """

    vel: float = aceleracion(masa, radio, angulo)*tiempo
    return masa*vel**2/2.0
```

Ejemplo (cont.)

```
def e_cinetica_rot(masa: float, radio: float, angulo: float,
                   tiempo: float) -> float:
    """
    Calcula la energía cinética de rotación,
    del objeto transcurrido el tiempo indicado (Julios)

    Args:
        masa: masa de la esfera, en kg
        radio: radio de la esfera, en m
        angulo: angulo del plano inclinado, en grados
        tiempo transcurrido, en s
    Returns:
        la energía cinética de rotación de la esfera en J
    """

    # Observar que el cálculo del momento de inercia aparece
    # repetido. Deberíamos implementarlo con una función
    momento_inercia: float = 2.0*masa*radio**2/5.0
    vel_angular: float = (aceleracion(masa, radio, angulo) *
                          tiempo/radio)
    return momento_inercia*vel_angular**2/2.0
```

Ejemplo (cont.)

```python
def main():
    """
    Programa principal que muestra para una esfera en
    un plano inclinado la distancia y energías a los
    cuatro segundos
    """

    # Crear el sistema plano-esfera
    # Ángulo 30 grados, esfera de 1.5 kg y r=0.2 m
    masa: float = 1.5
    radio: float = 0.2
    angulo: float = 30

    # Mostrar resultados con 3 decimales
    # Ponemos ":.3f" dentro de las {} para indicar 3 decimales
    print("Dist. a los 4 seg: " +
          f"{distancia(masa, radio, angulo, 4.0):.3f} m")
    print("E. C. tras. a 4 seg: " +
         f"{e_cinetica_tras(masa, radio, angulo, 4.0):.3f} J")
    print("E. C. rot a 4 seg: " +
          f"{e_cinetica_rot(masa, radio, angulo, 4.0):.3f} J")
```

Números aleatorios

El paquete `random` incluye operaciones para generar números aleatorios

Las más frecuentes:

`random.seed(), random.seed(i)`	inicializa el generador de números aleatorios con el reloj del sistema o con el número entero `i` ¡Hacer solo una vez!
`random.random()`	retorna un número aleatorio entre 0 y casi 1
`random.randint(start,stop)`	retorna un número aleatorio entero entre `start` y `stop` (incluidos)
`random.uniform(start,stop)`	retorna un número aleatorio real entre `start` y `stop` (incluidos)

Ejemplo con números aleatorios

En este ejemplo producimos una apuesta aleatoria para la bonoloto

- seis números del 1 al 49

```python
# -*- coding: utf-8 -*-
"""
Producimos una apuesta aleatoria para la bonoloto

Son seis números aleatorios del 1 al 49
Para un funcionamiento correcto habría que eliminar repeticiones

@author: Michael
@date  : Feb 2024
"""

import random
```

Ejemplo (cont.)

```
def main():
    """
    Muestra en pantalla seis números aleatorios entre 1 y 49
    """

    # Establecemos una semilla aleatoria
    random.seed()

    # Mostramos el número, una coma y sin salto de línea
    print(f"{random.randint(1, 49)}, ", end="")

    # Repetimos 4 veces más
    for _ in range(4):
        print(f"{random.randint(1, 49)}, ", end="")

    # La última vez no ponemos la coma
    print(random.randint(1, 49))
```


© Michael González Harbour, 28/feb/24

A observar en el ejemplo

Importancia de poner la semilla variable

- Si ponemos una fija, siempre saldrá la misma apuesta

Para hacer un `print` sin el salto de línea: `print(x, end="")`

Instrucción **`for`** para repetir unas instrucciones un número de veces determinado

- las instrucciones a repetir se especifican con el sangrado

El programa propuesto es limitado, ya que puede salir varias veces el mismo número, invalidando la apuesta

- Más adelante veremos cómo eliminar *repeticiones* de los números obtenidos, guardándolos en una lista o, mejor, en un *conjunto*

2.7. Variables y paso de parámetros

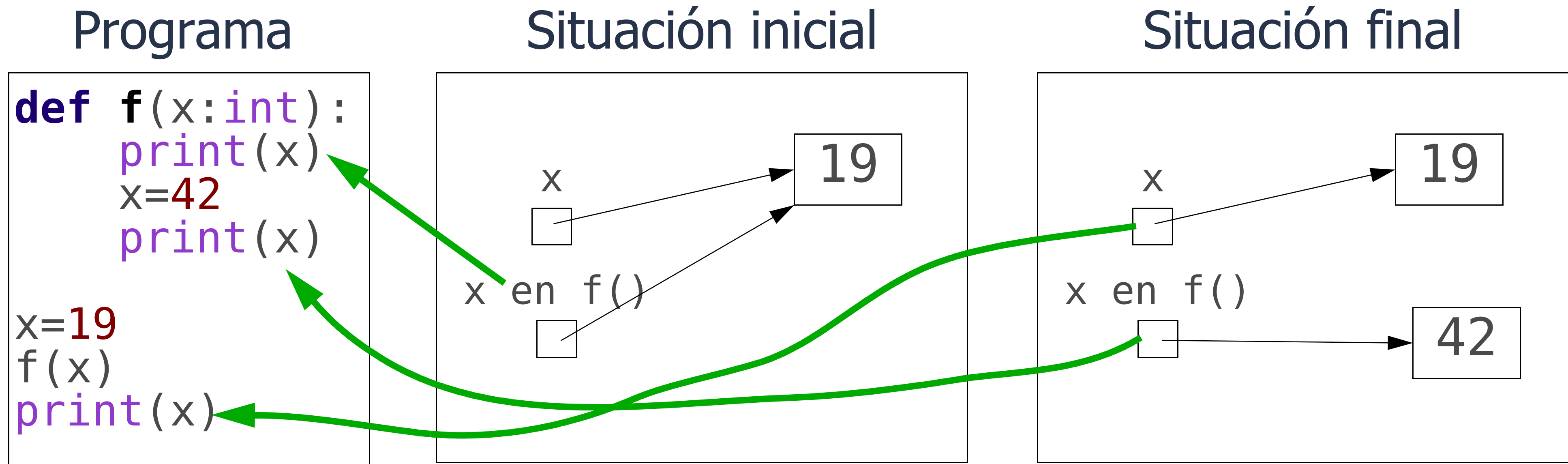
Como todas las variables, los parámetros de una función son referencias a objetos

- si el objeto original es *inmutable*, la función no lo puede cambiar
  - si se asigna otro objeto al parámetro, el original no cambia
  - por ejemplo números, booleanos, strings o tuplas
- pero si el objeto original es *mutable*, la función puede cambiarlo
  - el original cambia
  - por ejemplo, las listas

© Michael González Harbour, 28/feb/24

Ejemplo de paso de parámetros

Disponemos de este programa



La salida obtenida es

```
19
42
19
```

2.8 Listas y tuplas

La posibilidad de definir secuencias de objetos en la propia sintaxis de Python es una de sus características notables. Por ejemplo:

- tuplas

```
t=(1, 3, 7, "pepe")
```

- listas

```
l=[1, 3, 7, "pepe"]
```

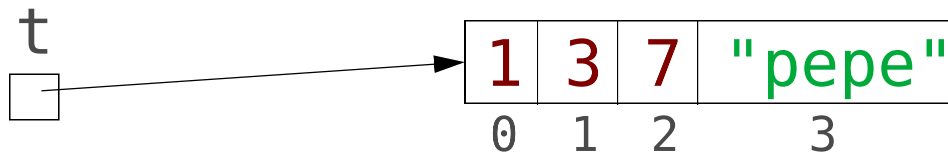
Tipos de secuencias

tipo	descripción	mutable
tuple	secuencia de elementos heterogéneos	no
list	secuencia de elementos heterogéneos	sí
str	secuencia de caracteres	no
range	secuencia de enteros seguidos	no

Numeración de los elementos

Las casillas de las secuencias se numeran comenzando por cero

- Llamamos *índice* a la numeración de las casillas



Puede haber listas y tuplas vacías

```
tupla_vacia=()
lista_vacia=[]
```

O de un solo elemento

```
t_uno = (3,) #observar la coma
l_uno = [4]
```

Operaciones comunes a las secuencias

Las listas y tuplas tienen las mismas operaciones que hemos visto para los strings; se muestran ejemplos con la tupla `t` y la lista `l` (página 75)

Operación	Descripción	Ejemplo	Resultado
`x in s`	`True` si la secuencia contiene a `x`	`"juan" in t`	`False`
`x + y`	Concatenar las dos secuencias	`(10,11)+(12,)`	`(10,11,12)`
`s * n`	Concatenar `s` a sí mismo `n` veces	`('a',)*3`	`('a','a','a')`
`s[i]`	Elemento `i`-ésimo de la secuencia	`l[1]`	`3`
`s[i:j]`	Rodaja de la secuencia, entre el índice `i` (incluido) y `j` (excluido)	`l[1:3]`	`[3, 7]`
`len(s)`	Número de elementos de `s`	`len(l)`	`4`
`min(s)`	Mínimo de los elementos de `s`	`min((1,5,3))`	`1`
`max(s)`	Máximo de los elementos de `s`	`max((1,5,3))`	`5`
`s.count(x)`	Número de ocurrencias de `x` en `s`	`l.count(2)`	`0`

© Michael González Harbour, 28/feb/24

Notación para obtener un elemento o un fragmento de una secuencia

En los fragmentos se puede *omitir* uno de los índices

Operación	Descripción	Ejemplo	Resultado
`s[i:]`	Rodaja de la secuencia, entre el índice `i` (incluido) y el final de la secuencia	`l[2:]`	`[7, "pepe"]`
`s[:j]`	Rodaja de la secuencia, entre el primer elemento y `j` (excluido)	`t[:3]`	`(1,3,7)`

Asimismo, para secuencias y fragmentos podemos usar *índices negativos*, que cuentan las casillas *desde el final*

Operación	Descripción	Ejemplo	Resultado
`s[-1]`	Último elemento	`l[-1]`	`"pepe"`
`s[-3:]`	Los tres últimos elementos	`t[-3:]`	`(3,7, "pepe")`

© Michael González Harbour, 28/feb/24

Operaciones con secuencias mutables

Las listas tienen además estas operaciones. Se ponen ejemplos con `l`

Operación	Descripción	Ejemplo	Resultado
`s[i]=x`	Reemplaza la casilla `i` de `s` por `x`	`l[1]='a'`	`[1,'a',7,'pepe']`
`s[i:j]=t`	Se reemplaza la rodaja `i:j` de `s` por los contenidos de `t`	`l[1:3]=(2,5)`	`[1,2,5,'pepe']`
`del(s[i:j])`	Se elimina la rodaja `i:j` de `s`	`del(l[1:3])`	`[1,'pepe']`
`s.append(x)`	Añade `x` al final de `s`	`l.append(8)`	`[1,3,7,'pepe',8]`
`s.clear()`	Borra todos los elementos de `s`	`l.clear()`	`[]`
`s.insert(i,x)`	Inserta `x` en `s`, en la casilla `i`	`l.insert(2,'b')`	`[1,3,'b',7, 'pepe']`

© Michael González Harbour, 28/feb/24

Operaciones con secuencias mutables (cont.)

Operación	Descripción	Ejemplo	Resultado
`s.remove(x)`	Elimina la primera aparición del elemento x	`l.remove("pepe")`	`[1,3,7]`
`s.reverse()`	Invierte los elementos de s	`l.reverse()`	`['pepe',7,3,1]`

© Michael González Harbour, 28/feb/24

Rangos

Son secuencias de números enteros

```
range(10) # 0,1,2,3,4,5,6,7,8,9
range(2,10) # 2,3,4,5,6,7,8,9
range(2,11,2) # 2,4,6,8,10, de dos en dos
```

Se usan para hacer bucles

```
for i in range(10): # i toma los valores de 0 a 9
    # las instrucciones se repiten 10 veces
    instrucciones
```

o crear listas o tuplas

```
l = list(range(10))  # [0,1,2,3,4,5,6,7,8,9]
t = tuple(range(10)) # (0,1,2,3,4,5,6,7,8,9)
```

Ejemplo con lista, tupla y rango

```
# -*- coding: utf-8 -*-
"""
Trabaja con los nombres y días de los meses

@author: Michael
@date  : Feb 2024
"""
```

Ejemplo con lista, tupla y rango (cont)

```
def main():
    """
    Muestra en pantalla los días de cada mes de 2024,
    con sus nombres
    """

    nombre_mes: list[str] = \
        ["enero", "febrero", "marzo", "abril", "mayo", "junio",
         "julio", "agosto", "septiembre", "octubre",
         "noviembre", "diciembre"]

    dias_mes: tuple[int, ...] = (31, 29, 31, 30, 31, 30, 31,
                                 31, 30, 31, 30, 31)

    print("Días de cada mes de 2024:")
    for i in range(12):
        print(f"{nombre_mes[i]}: {dias_mes[i]}")
```

continuación de línea

no necesita continuación de línea

© Michael González Harbour, 28/feb/24

A observar

Cuando una línea es muy larga se puede dividir en dos o más

- el carácter `\` sirve como continuación de línea
  - la línea posterior lleva un nivel de sangrado más, como en el ejemplo anterior
- tal como vimos en otros ejemplos, cuando queremos separar la línea dentro de `()` o `[]` podemos hacerlo sin más, sin usar el carácter `\`
  - en ese caso, se sangra la nueva línea justo debajo del contenido del paréntesis o corchete, como en el ejemplo anterior
- no podemos separar la línea en mitad de un nombre, o de un literal o de un string que no sea multilínea

© Michael González Harbour, 28/feb/24

Apéndice: Comprobación de anotaciones de tipo

Las anotaciones de tipo tienen como objetivo ayudar a la legibilidad y consistencia del código

- pero el intérprete Python no las comprueba

La herramienta `mypy` permite comprobar que el programa no viola los tipos anotados

Instalación desde el terminal `Anaconda prompt`:

```
conda install mypy
```

Uso desde el terminal `Anaconda prompt`:

```
mypy --ignore-missing-imports mi_fichero.py
```


© Michael González Harbour, 28/feb/24

Apéndice: Integración de mypy con Spyder

Spyder puede mostrar de forma automática errores de tipificación

- A fecha de hoy, en Windows no siempre funciona bien

Para instalar esta opción desde el terminal `Anaconda prompt`:

```
conda install -c conda-forge pylsp-mypy
```

A continuación crear un fichero de configuración llamado `.mypy.ini` (puede quedar oculto) con estos contenidos:

```
[mypy]
ignore_missing_imports = True
check_untyped_defs = True
```

En Windows poner el fichero en `C:\Usuarios\nombre_usuario`

- siendo `nombre_usuario` el nombre del usuario

En Linux y Mac poner el fichero en el directorio del usuario

© Michael González Harbour, 28/feb/24