

# Capítulo 2



Python

# Listas



- Permiten indexar elementos
  - Índices son números correlativos comenzando de cero
  - Pueden tener elementos de distinto tipo

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
```

```
>>> squares[0]  # indexing returns the item
1
>>> squares[-1]
25
>>> squares[-3:]  # slicing returns a new list
[9, 16, 25]
```

# Listas



- Podemos concatenar listas

```
>>> squares + [36, 49, 64, 81, 100]  
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

- A diferencia de string, listas son mutables:

```
>>> cubes = [1, 8, 27, 65, 125] # something's wrong here  
>>> 4 ** 3 # the cube of 4 is 64, not 65!  
64  
>>> cubes[3] = 64 # replace the wrong value  
>>> cubes  
[1, 8, 27, 64, 125]
```

# Listas



- Agregar elementos a la lista: método `append`

```
>>> cubes.append(216)  # add the cube of 6
>>> cubes.append(7 ** 3)  # and the cube of 7
>>> cubes
[1, 8, 27, 64, 125, 216, 343]
```

- Otros métodos de listas:
  - <https://docs.python.org/3/tutorial/datastructures.html>

# Listas



- Trabajando con porciones de la lista:

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> # replace some values
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> # now remove them
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']
>>> # clear the list by replacing all the elements with an empty list
>>> letters[:] = []
>>> letters
[]
```

# Listas



- Podemos tener listas de listas

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
```

# Python



- Ejemplo serie de Fibonacci

```
>>> # Fibonacci series:
... # the sum of two elements defines the next
... a, b = 0, 1
>>> while a < 10:
...     print(a)
...     a, b = b, a+b
...
0
1
1
2
3
5
8
```

# Condicionales



```
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
...
More
```



# Ciclo for



```
>>> # Measure some strings:
... words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print(w, len(w))
...
cat 3
window 6
defenestrate 12
```

# Función range



```
>>> for i in range(5):  
...     print(i)  
...  
0  
1  
2  
3  
4
```

```
range(5, 10)  
5, 6, 7, 8, 9
```

```
range(0, 10, 3)  
0, 3, 6, 9
```

```
range(-10, -100, -30)  
-10, -40, -70
```

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']  
>>> for i in range(len(a)):  
...     print(i, a[i])  
...  
0 Mary  
1 had  
2 a  
3 little  
4 lamb
```

# Funciones



```
>>> def fib(n):      # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print(a, end=' ')
...         a, b = b, a+b
...     print()
...
>>> # Now call the function we just defined:
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

# Funciones: retorna una lista



```
>>> def fib2(n): # return Fibonacci series up to n
...     """Return a list containing the Fibonacci series up to n."""
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a)      # see below
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100)      # call it
>>> f100                  # write the result
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

# Funciones: argumentos con valores



```
def ask_ok(prompt, retries=4, reminder='Please try again!'):
    while True:
        ok = input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise ValueError('invalid user response')
        print(reminder)
```

```
ask_ok('Do you really want to quit?')
```

```
ask_ok('OK to overwrite the file?', 2)
```

```
ask_ok('OK to overwrite the file?', 2, 'Come on, only yes or no!')
```

# Funciones: documentación



```
>>> def my_function():  
...     """Do nothing, but document it.  
...  
...     No, really, it doesn't do anything.  
...     """  
...     pass  
...  
>>> print(my_function.__doc__)  
Do nothing, but document it.  
  
    No, really, it doesn't do anything.
```

# Diccionarios



- Similares a “arreglos asociativos” de otros lenguajes
  - Conjunto de pares llave - valor
  - Indexados por una llave en lugar de un rango de números enteros
  - Si se agrega un valor con una llave existente, el valor antiguo se pierde
  - Generará un error extraer un valor con una llave inexistente

# Diccionarios



```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'jack': 4098, 'sape': 4139, 'guido': 4127}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'jack': 4098, 'guido': 4127, 'irv': 4127}
>>> list(tel)
['jack', 'guido', 'irv']
>>> sorted(tel)
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```



# Diccionarios



- Recorriendo el diccionario:

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.items():
...     print(k, v)
...
gallahad the pure
robin the brave
```

# Excepciones



- Corresponden a errores detectados durante la ejecución del programa

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

# Excepciones



- Gestión de excepciones

```
>>> while True:
...     try:
...         x = int(input("Please enter a number: "))
...         break
...     except ValueError:
...         print("Oops! That was no valid number. Try again...")
... 
```

- Un try puede tener más de un except
  - Un except puede ser para varias excepciones

```
... except (RuntimeError, TypeError, NameError):
...     pass
```

# Excepciones



- Un except que sirve como comodín para cualquier excepción:

```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except OSError as err:
    print("OS error: {0}".format(err))
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error:", sys.exc_info()[0])
    raise
```

# Excepciones: try, except, else



```
for arg in sys.argv[1:]:  
    try:  
        f = open(arg, 'r')  
    except OSError:  
        print('cannot open', arg)  
    else:  
        print(arg, 'has', len(f.readlines()), 'lines')  
        f.close()
```

# Excepciones



- Lanzando excepciones con raise:

```
>>> try:
...     raise NameError('HiThere')
... except NameError:
...     print('An exception flew by!')
...     raise
...
An exception flew by!
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
NameError: HiThere
```


# Excepciones



- Acciones de limpieza: `finally`

```
>>> try:
...     raise KeyboardInterrupt
... finally:
...     print('Goodbye, world!')
...
Goodbye, world!
KeyboardInterrupt
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
```

```
>>> def bool_return():
...     try:
...         return True
...     finally:
...         return False
...
>>> bool_return()
False
```



```
>>> def divide(x, y):
...     try:
...         result = x / y
...     except ZeroDivisionError:
...         print("division by zero!")
...     else:
...         print("result is", result)
...     finally:
...         print("executing finally clause")
...
```

```
>>> divide(2, 1)
result is 2.0
executing finally clause
```

```
>>> divide(2, 0)
division by zero!
executing finally clause
```

```
>>> divide("2", "1")
executing finally clause
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in divide
```

```
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```