Python 3

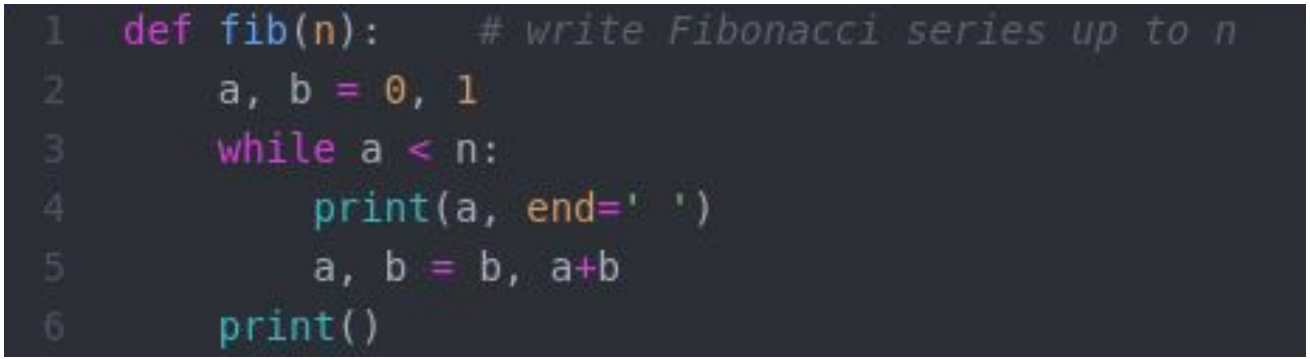
> def run():

Head first!

```
1  def beautify_text(text):  # function header
2      #  this code block is the function body
3      beautified = f'★·.·´¯`·.·★ [ {text} ] ★·.·´¯`·.·★'
4      return beautified  # return value
5
6
7  def run():
8      print(beautify_text('Hello World'))
9
10
11 run()
12
13 ★·.·´¯`·.·★ [ Hello World ] ★·.·´¯`·.·★
14
15
```

Declaración de funciones

La palabra reservada def comienza la definición de una función. Debe ser seguida por el nombre de la función y sus parámetros.



Valores de retorno

La sentencia return indica el retorno de la función. Si una función no hace uso de esta sentencia devolverá el valor None.

```
>>> def add_one(n):
...     return n + 1
...
>>> print(add_one(9))
10
>>>
>>> def get_none():
...     pass
...
>>> get_none()
>>> get_none() is None
True
```

Las funciones pueden devolver cualquier tipo de objeto.

Devolver un objeto de tipo tuple puede ser útil para obtener más de un valor de retorno de una función.

```
>>> def get_max_min(l_int):
...     return (max(l_int), min(l_int))
...
>>> get_max_min([0, 7, 5, 5])
(7, 0)
>>> max, min = get_max_min([0, 7, 5, 5])
>>> max
7
>>> min
0
```

Modificando argumentos

Las funciones reciben como argumento referencias a objetos. Los **mutables** que se pasen a la función pueden ser alterados pero los **inmutables** no.

```python
def clear_list(l):
    # l is altered but it references
    # the same list
    l[:] = []


l = [1, 2, 3]
clear_list(l)
print(l)
```

```
[]
```

```python
def add_one(i):
    # this does not work because i will
    # reference a new object
    i += 1


x = 5
add_one(x)
print(x)
```

```
5
```

```python
def clear_list(l):
    # this is wrong because l will
    # reference a new list
    l = []


l = [1, 2, 3]
clear_list(l)
print(l)
```

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

Variables dentro de funciones

La ejecución de una función introduce una nueva tabla de símbolos. Todas las asignaciones de variables en una función se hacen a nivel local. Las referencias a variables tratan de resolverse primero localmente, luego en las tablas de símbolos de funciones que contienen a la función actual, luego globalmente y por último en la tabla built-in.

```
x = 5

def print_x():
    print(f'Global reference: {x}')  # x can be referenced

def alter_x():
    x = 0  # this will introduce a new local symbol
    print(f'Local reference: {x}')

print_x()
alter_x()
print(f'Global reference (unaltered): {x}')
```

```
Global reference: 5
Local reference: 0
Global reference (unaltered): 5
```

Referencias a funciones

La definición de una función introduce un nuevo símbolo en la tabla correspondiente. El valor de la función es un tipo que es reconocido por el interprete como una función definida por el usuario (function). Se puede asignar una referencia a una función a una variable.

```
1  def say_hello(to):  # say_hello references a function
2      print(f'Hello {to}!')
3
4
5  greet = say_hello  # greet references the same function as say_hello
6
7  greet('Joe')  # call the funcion using its new name
```

Valores de parámetros por default

Se pueden declarar funciones que reciban argumentos opcionales, y en el caso de ser omitidos asignarles un valor por default.

```
1  def print_and_repeat(text, times=3, begin='==> '):
2      for i in range(times):
3          print(f'{begin}{text}')
4
5
6  print_and_repeat('Called with one argument.')
7  print_and_repeat('Called with two arguments.', 5)
8  print_and_repeat('Called with all arguments.', 5, 'LINE: ')
```

Los valores por default solo se evalúan una vez.

Keyword arguments

Las funciones pueden ser llamadas utilizando **keyword arguments**. Estos toman la forma kwarg=value y tienen que ser especificados luego de los **positional arguments**.

```
def print_and_repeat(text, times=3, begin='==> '):
    for i in range(times):
        print(f'{begin}{text}')


print_and_repeat('Valid.', begin='~> ')  # valid call
print_and_repeat(begin='~> ', text='Valid.')  # also a valid call

print_and_repeat(begin='~> ')  # invalid call (text is required)
print_and_repeat('Test.', times=5, '~> ')  # invalid call
```

Cantidad de argumentos no determinada (I)

Declarando un parámetro de la forma `*name` se puede recibir una cantidad no determinada de **positional arguments** que serán contenidos dentro de un `tuple`.

Utilizando `**name` se puede recibir una cantidad no determinada de **keyword arguments** que serán contenidos dentro de un `dict`.

Si se especifica `**name` debe hacerse al final de la lista de argumentos.

```python
def multi_print(*args, **kwargs):
    print('Positional arguments:')
    for arg in args:  # args is a tuple
        print(f'\t{arg}')
    print('Keyword arguments:')
    for k, v in kwargs.items():  # kwargs is a dict
        print(f'\t{k} => {v}')


multi_print('pos1', 'pos2', 'pos3', a='kw1', b='kw2', c='kw3')
```

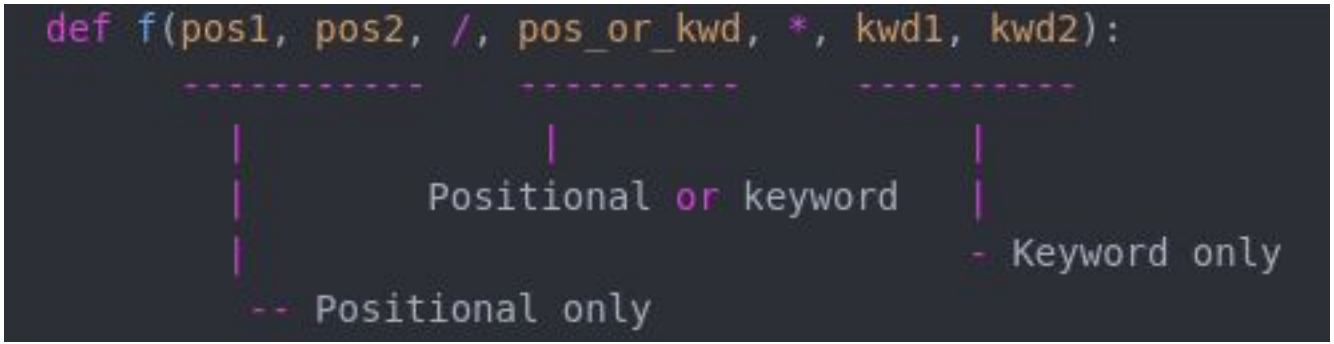
```
Positional arguments:
        pos1
        pos2
        pos3
Keyword arguments:
        a => kw1
        b => kw2
        c => kw3
```

Cantidad de argumentos no determinada (II)

Los parámetros / y *

El parámetro / fuerza a que los parámetros previos a él sean utilizados como **positional arguments.**

El parámetro * fuerza a que los parámetros posteriores a él sean utilizados como **keyword arguments.**



```python
# declaration of role constants
ROLE_ACCOUT_MANAGER = 'account_manager'
ROLE_SALESMAN = 'salesman'

# users DB
USERS = [
    {'username': 'joe', 'name': 'Joe', 'role': ROLE_ACCOUT_MANAGER},
    {'username': 'peter', 'name': 'Peter', 'role': ROLE_SALESMAN},
    {'username': 'jane', 'name': 'Jane', 'role': ROLE_ACCOUT_MANAGER}
]


# this function will return a list of users that are account managers
def get_account_managers():

    # this is an inner function
    def is_account_manager(user):
        return user['role'] == ROLE_ACCOUT_MANAGER

    # filter receives is_account_manager function as argument
    return list(filter(is_account_manager, USERS))
```

Funciones como argumentos y funciones internas

Expresiones Lambda

La palabra reservada lambda se utiliza para crear pequeñas funciones anónimas que evalúan una **única expresión** y retornan su valor.

Semánticamente son equivalentes a una función regular.

```
>>> lambda x, y: x + y
<function <lambda> at 0x7f5c0a886758>
>>> (lambda x, y: x + y)(10, 5)
15
```

```
# add two lists using map and lambda

l1 = [1, 2, 3]
l2 = [4, 5, 6]

result = map(
    lambda x, y: x + y,
    l1,
    l2
)

print(list(result))
```

```
[5, 7, 9]
```

Built-in functions

Some built-in functions (I)

`abs(x)`
Devuelve el valor absoluto de `x` (`float` o `int`).

`all(iterable)`
Devuelve `True` si todos los elementos evalúan como `True` o `iterable` está vacío.

`any(iterable)`
Devuelve `True` si alguno de los elementos evalúa como `True`. Si `iterable` está vacío devuelve `False`.

`chr(i)`
Devuelve un `string` con el carácter de Unicode correspondiente a `i`.

`divmod(a, b)`
Devuelve un `tuple` (cociente, resto) de la división entera `a / b`.

Some built-in functions (II)

`filter(function, iterable)`
Devuelve un iterable con los elementos de `iterable` que cumplen con el filtro `function(item)`.

`getattr(object, name[, default])`
Devuelve el valor del atributo `name` de `object`. Si `name` no existe devuelve `default` (si fue especificado) o lanza `AttributeError`.

`hasattr(object, name)`
Devuelve `True` si `object` tiene el atributo `name`.

`help([object])`
Comienza la ayuda interactiva si `object` no es especificado. En caso contrario muestra la documentación de `object`.

Some built-in functions (III)

`isinstance(object, classinfo)`
Devuelve `True` si `object` es una instancia de `classinfo` (directa indirecta o virtualmente). Si se pasa un `tuple` como `classinfo` se devuelve `True` si `object` es instancia de alguna de esas clases.

`issubclass(class, classinfo)`
Devuelve `True` si `class` es una subclase de `classinfo`. Una clase es considerada subclase de sí misma. También `classinfo` puede ser `tuple`.

`map(function, iterable, ...)`
Devuelve un iterable que aplica `function(item)` a cada elemento de `iterable`. Si se pasa más de un iterable entonces `function` debe tomar tantos argumentos como iterables se pasen (en este caso la iteración se termina con el iterable más corto).

Some built-in functions (IV)

`max(iterable, *[, key, default])`
`max(arg1, arg2, *args[, key])`
Devuelve el máximo valor de un iterable o múltiples argumentos. Opcionalmente se puede especificar `key(item)` que tiene que devolver el valor usado para las comparaciones. Si `iterable` está vacío se lanza `ValueError` o se devuelve `default` (si fue especificado). Si dos objetos son máximos entonces se devuelve el primero de los dos.

`min(iterable, *[, key, default])`
`min(arg1, arg2, *args[, key])`

`ord(c)`
Devuelve el código Unicode (`int`) del caracter `c`.

`reversed(seq)`
Devuelve un iterable invertido de `seq`.

Some built-in functions (V)

`round(number[, ndigits])`
Devuelve un valor numérico redondeado. Opcionalmente se puede especificar `ndigits` para redondear a una cierta cantidad de decimales. Si `number` es equidistante a dos valores redondeados entonces se opta por el valor par.

`sorted(iterable, *, key=None, reverse=False)`
Devuelve una `list` ordenada de los elementos de `iterable`. Se puede especificar `key(item)` para especificar otro valor de orden.

`sum(iterable, /, start=0)`
Suma `start` y los elementos de `iterable` de izquierda a derecha. También se pueden “sumar” valores no numéricos.

`zip(*iterables)`
Devuelve un iterador de tuplas. La tupla i-esima contiene el elemento i-esimo de cada iterable. Se devuelven tantas tuplas como elementos del iterable más corto.

code()

Ejercicios 4 (I)

- Crear una función que reciba una cantidad no definida de `string` y las imprima separándolas con un separador que también es pasado como argumento. Por ejemplo `super_print("String", "Long String", sep="|-.+:")` deberá imprimir:

```
|-.+:|-.+:|
String
|-.+:|-.+:|
Long String
|-.+:|-.+:|
```

Ejercicios 4 (II)

- Escribir un programa que permita cargar usuarios de un sistema. Cada usuario debe tener un id (entero), username, nombre y apellido. Se debe validar que el id y el username sean únicos, y además que el username tenga al menos 3 caracteres de longitud y solo contenga letras en minúscula.
  Al finalizar el ingreso de los usuarios el programa debe imprimir un listado de usuarios ordenados por username de manera ascendente.

Bibliografía

- Python docs
- How to Think Like a Computer Scientist: Learning with Python