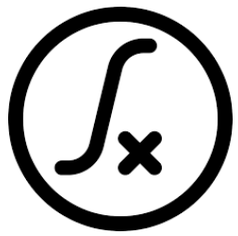
DataJam

BEGINNERS

Functions

Concepts & Introduction



Context : Functions

- Functions → reusable code
- Transform inputs into outputs
- 2 functions cannot have the same name
- Document your functions

Built-in vs Custom Functions

- str(x)
- len(x)
- type(x)
- ~~os.isfile(x)~~

- print_my_stuff(x)
- count_this(x)
- is_really(x)

In Python 3.6 (latest version), there are 68 built-in functions.
Check: https://www.programiz.com/python-programming/methods/built-in

DataJam

Functions basics

A function is a set of statements that take inputs, do some specific computation and produces output.
The idea is to put some commonly or repeatedly done task together and make a function,
so that instead of writing the same code again and again for different inputs, we can call the function.

```
# A simple Python function to check
# whether x is even or odd
def evenOdd( x ):
    if (x % 2 == 0):
        print "even"
    else:
        print "odd"

# Driver code
evenOdd(2)
evenOdd(3)
```

Output:

```
even
odd
```

```
def myFun(x):

    # After below line link of x with previous
    # object gets broken. A new object is assigned
    # to x.
    x = [20, 30, 40]

# Driver Code (Note that lst is not modified
# after function call.
lst = [10, 11, 12, 13, 14, 15]
myFun(lst);
print(lst)
```

Output:

```
[20, 11, 12, 13, 14, 15]
```

Functions (docstrings)

- Docstring → Get info about the function definition & role

A **docstring** is a string literal that occurs as the first statement in a module, function, class, or method definition.
Such a **docstring** becomes the __doc__ special attribute of that object.
All modules should normally have **docstrings**,
and all functions and classes exported by a module should also have **docstrings**.

BUILD FUNCTIONS (TOY EXAMPLES)

In [330]:
```
def my_first_method():
    """
    this is the docstring info... for an useless function
    """
    print("hello world froom python")
```

In [331]:
```
print(my_first_method.__doc__)
```

```
    this is the docstring info... for an useless function
```

In [332]:
```
def my_third_function(x="Assan"):
    """
    :param my_parameter_1: this is your first param
    :param my_parameter_2: this is your second param
    :returns: customized greeting
    """

    print("Hi!"+ " " + str(x) + "this is a classy hello world" )
```

In [333]:
```
print(my_third_function.__doc__)
```

```
    :param my_parameter_1: this is a first param
    :param my_parameter_2: this is a second param
    :returns: customized greeting
```

DataJam

Functions (*args,**kwargs)

In [335]:
```
def print_some_stuff(x):
    """
    take an unique argument and print it
    """
    print(x)
```

In [337]:
```
# in the case of 1 argument , we do respect the function signature ->it is going to work
print_some_stuff("datajam is legit")
```

```
datajam is legit
```

In [338]:
```
# in the case of 2 arguments , we don't respect the function signature ->it is going to fail
print_some_stuff("datajam is legit", "dsti is too")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-338-0da82d94f044> in <module>
----> 1 print_some_stuff("datajam is legit", "dsti is too")

TypeError: print_some_stuff() takes 1 positional argument but 2 were given
```

In [379]:
```
def print_some_stuff_plus(*args):
    """
    take an iterable of all the arguments supplied (whatever number of arguments supplied)
    """

    my_string= []
    for el in args:
        my_string.append(el)
    print( " and ".join(my_string))
```

In [380]:
```
# in the case of 2 arguments - because of the iterable  -> it is going to work
print_some_stuff_plus("datajam is legit","dsti is too")
```

```
datajam is legit and dsti is too
```

In [381]:
```
# in the case of k arguments - because of the iterable  -> it is going to work
print_some_stuff_plus("datajam is legit","dsti is too", "you are legit as well")
```

```
datajam is legit and dsti is too and you are legit as well
```

The single asterisk (*) is used to unpack iterables (list or strings for ex)

The two asterisks (**) is used to unpack dictionaries

Read more @: *https://realpython.com/python-kwargs-and-args/*

DataJam

Functions (*args,**kwargs)

```
In [374]: def concatenate_stuff(x="truc", y="muche"):
              return(x + ","+ y)

In [375]: concatenate_stuff(x="machin",y="bidule")

Out[375]: 'machin,bidule'

In [376]: concatenate_stuff(x="machin",y="bidule",z="truc")

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-376-4ba1c5cb84de> in <module>
----> 1 concatenate_stuff(x="machin",y="bidule",z="truc")

TypeError: concatenate_stuff() got an unexpected keyword argument 'z'

In [377]: def concatenate_stuff_plus(**kwargs):
              my_returned_value = []
              for val in kwargs.values():
                  my_returned_value.append(str(val))
              return(",".join(my_returned_value))

In [378]: concatenate_stuff_plus(x="machin",y="bidule",z="truc")

Out[378]: 'machin,bidule,truc'
```

The single asterisk (*) is used to unpack iterables (list or strings for ex)

The two asterisks (**) is used to unpack dictionaries

Read more @: *https://realpython.com/python-kwargs-and-args/*