15.FunctionsInpython

May 6, 2020

[1]:
```
data = [ 'hello', 'hi', 'bye', 'good', 'bye', 'let', 'begin', 'show']
```

[2]:
```
for i, item in enumerate(data):
    print(f"{8-i}. {item}")
else:
    print("\nTell me the output ? ")
```

```
8. hello
7. hi
6. bye
5. good
4. bye
3. let
2. begin
1. show

Tell me the output ?
```

zip

[3]:
```
from random import randint
```

List Comprehension

[6]:
```
l = [ var for var in range(1, 11)]
print(l)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[7]:
```
l = [ item for item in range(1, 21) if item % 2 == 0 ]
```

[9]:
```
l
```

[9]:
```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

[17]:
```
data = [
    (i, j, z)
            for i in range(2) # i = 0
            for j in range(2) # j = 1
```

```
            for z in range(2)
]
```

[18]:
```
print(data)
```

```
[(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0),
(1, 1, 1)]
```

[19]:
```
from random import randint
```

[20]:
```
x = [ 1, 2, 3, 4, 5 ]
y = [ randint(1, 10) for var in range(5)]
```

[21]:
```
print(x)
print(y)
```

```
[1, 2, 3, 4, 5]
[2, 10, 6, 5, 3]
```

[22]:
```
r = []
i = 0
while i < len(x):
    e1 = x[i] ** 2
    e2 = y[i] ** 2
    s = e1 + e2
    r.append(s)
    i += 1
```

[23]:
```
print(r)
```

```
[5, 104, 45, 41, 34]
```

[27]:
```
r = []
for i in range(len(x)): # 0, 1, 2, 3, 4
    r.append(x[i]**2+y[i]**2)
```

[28]:
```
r
```

[28]:
```
[5, 104, 45, 41, 34]
```

[29]:
```
print(x)
```

```
[1, 2, 3, 4, 5]
```

[30]:
```
print(y)
```

```
[2, 10, 6, 5, 3]
```

[32]:
```
#help(zip) # detail help about func or class
print(zip.__doc__) # __doc__ represents doc-string of a function
```

```
zip(iter1 [,iter2 […]]) --> zip object

Return a zip object whose .__next__() method returns a tuple where
the i-th element comes from the i-th iterable argument.  The .__next__()
method continues until the shortest iterable in the argument sequence
is exhausted and then it raises StopIteration.
```

[33]:
```
z = zip(x, y)
```

[35]:
```
print(z) # iterable objects, combination, collection
# memory efficient objects which does store data instead store state (some␣
 ↪variable)
# basically they generate data on demand thus we call them generators or␣
 ↪iterators
```

```
<zip object at 0x0000021F0EC3CEC8>
```

```
range, enumerate, zip, map, --> generators
```

[36]:
```
## next function is used get data from generators type objects
```

[37]:
```
print(x)
```

```
[1, 2, 3, 4, 5]
```

[38]:
```
print(y)
```

```
[2, 10, 6, 5, 3]
```

[39]:
```
z
```

[39]:
```
<zip at 0x21f0ec3cec8>
```

[40]:
```
next(z)
```

[40]:
```
(1, 2)
```

[41]:
```
next(z)
```

[41]:
```
(2, 10)
```

[42]:
```
next(z)
```

[42]:
```
(3, 6)
```

[43]:
```
next(z)
```

[43]:
```
(4, 5)
```

[44]:
```
next(z)
```

[44]:
```
(5, 3)
```

[45]:
```
next(z)
```

```
␣
↪---------------------------------------------------------------------------

       StopIteration                                Traceback (most recent call␣
↪last)

       <ipython-input-45-cf9ac561a401> in <module>
   ----> 1 next(z)


       StopIteration:
```

[46]:
```
print(x)
```

```
[1, 2, 3, 4, 5]
```

[47]:
```
print(y)
```

```
[2, 10, 6, 5, 3]
```

[48]:
```
print(*zip(x, y), sep='\n')
```

```
(1, 2)
(2, 10)
(3, 6)
(4, 5)
(5, 3)
```

[49]:
```
name  =  [ 'sachin', 'rajat', 'nidhi', 'kushal', 'manish', 'tanvi']
maths = [ 100, 90, 78, 65, 76, 90]
sci   =   [ 78, 76, 78, 56, 76, 88]
```

[50]:
```
#data = [ ('sachin', 100, 78), ('rajat', 90, 76), ....]
```

[51]:
```
for n, m, s in zip(name, maths, sci):
    print(n, m, s)
```

```
sachin 100 78
rajat 90 76
nidhi 78 78
kushal 65 56
manish 76 76
tanvi 90 88
```

[54]:
```
data = [ [n, m, s] for n, m, s in zip(name, maths, sci)]
```

[55]:
```
data
```

[55]:
```
[['sachin', 100, 78],
 ['rajat', 90, 76],
 ['nidhi', 78, 78],
 ['kushal', 65, 56],
 ['manish', 76, 76],
 ['tanvi', 90, 88]]
```

[56]:
```
s1 = 'python'
s2 = 'sachin yadav'
```

[57]:
```
print(*zip(s1, s2), sep='\n')
```

```
('p', 's')
('y', 'a')
('t', 'c')
('h', 'h')
('o', 'i')
('n', 'n')
```

[58]:
```
l1 = [ 'hello', 'world', 'how']
l2  = [ 'are', 'you', 'guys', 'let', 'start']
```

[59]:
```
for e1, e2 in zip(l1, l2):
    print(e1, e2)
```

```
hello are
world you
how guys
```

[61]:
```
x = [ randint(1, 10) for var in range(5) ]
y = [ randint(1, 10) for var in range(5)]
```

[62]:
```
print(x)
```

```
[8, 9, 7, 7, 5]
```

[63]:
```
print(y)
```

```
[10, 3, 9, 6, 4]
```

[64]:
```
r = [ e1**2+e2**2 for e1, e2 in zip(x, y) ]
print(r)
```

```
[164, 90, 130, 85, 41]
```

Functions

What is function ? why we use functions ?

```
functions are set of statements which are used to perform a specific task in programming and ca
```

Features of Function

```
* Code Reusability
* Easy Debugging
* Modular Programming
* Recursion & Dynamic Programming
* Make Coding Easy (Writing & Reading)
```

Builtin Functions –> which comes with language

User Defined Functions –> custom function to perform custom task

[65]:
```
import builtins
```

[66]:
```
print(dir(builtins))
```

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException',
'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning',
'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning',
'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning',
'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError',
'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError',
'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError',
'NameError', 'None', 'NotADirectoryError', 'NotImplemented',
'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning',
'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError',
'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration',
'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit',
'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError',
'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError',
'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError',
'Warning', 'WindowsError', 'ZeroDivisionError', '__IPYTHON__',
'__build_class__', '__debug__', '__doc__', '__import__', '__loader__',
'__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin',
'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod',
'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir',
```

```
'display', 'divmod', 'enumerate', 'eval', 'exec', 'filter', 'float', 'format',
'frozenset', 'get_ipython', 'getattr', 'globals', 'hasattr', 'hash', 'help',
'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len',
'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next',
'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'range', 'repr',
'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str',
'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

[ ]:

[67]:
```
func = [ f for f in dir(builtins) if f[0].islower() ]
print(func)
```

```
['abs', 'all', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray',
'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright',
'credits', 'delattr', 'dict', 'dir', 'display', 'divmod', 'enumerate', 'eval',
'exec', 'filter', 'float', 'format', 'frozenset', 'get_ipython', 'getattr',
'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance',
'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max',
'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print',
'property', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice',
'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

[69]:
```
print(len(func))
```

```
73
```

[70]:
```
abs(-123) # | mod |
```

[70]:
```
123
```

[73]:
```
abs(123.12)
```

[73]:
```
123.12
```

[74]:
```
cond = [ True, False, True, True]
```

[75]:
```
all(cond) # all works like and
```

[75]:
```
False
```

[76]:
```
all([ True, True, 'hi', 'hello'])
```

[76]:
```
True
```

[77]:
```
any([ True, False, 0, {}])
```

[77]:
```
True
```

[78]:
```
license()
```

```
See https://www.python.org/psf/license/
```

[79]:
```
credits()
```

```
    Thanks to CWI, CNRI, BeOpen.com, Zope Corporation and a cast of thousands
    for supporting Python development.  See www.python.org for more information.
```

built-in function you have to study

Custom Functions

Without Arguments Without Return Type

Without Arguement With Return Type

With Argument Without Return Type

With Argument With Return Type

[81]:
```
# this is calling
x = print() # Without Argument Without Return Type
print(x) #  With Argument Without Return Type
```

```
None
```

[82]:
```
name = input() # Without Argument with return type string
print(name)
```

```
45
45
```

[83]:
```
name = input("Name: ") # with argument with return type
print(name)
```

```
Name: sachin
sachin
```

how to create functions in python ?

Function is just a callable object

Syntax:

defination --> which store group of logical statements to perform a specific task

calling --> used to executes stored statements of a function

```
def func_name(arg1, arg2, ...): # arg1, arg2 --> formal parameters
```

```
    """
        doc string of function explaining everything about how to use this function
    """
    st-1
    st-2
    st-3
    ...
    st-n

    return value


value = func_name(para1, para2, ....) # para1, para2 --> actual parameters
```

[86]:
```
def hello(name):
    """
        hello(name) is a time pass function to explain functions.

        this is called doc-string of function
    """
    s = f"Welcome {name} to the world of functions in python."
    return s
```

[87]:
```
help(hello)
```

```
Help on function hello in module __main__:

hello(name)
    hello(name) is a time pass function to explain functions.

    this is called doc-string of function
```

[88]:
```
print(hello.__doc__)
```

```

        hello(name) is a time pass function to explain functions.

        this is called doc-string of function
```

[89]:
```
value = hello("Sachin Yadav")
```

[90]:
```
print(value)
```

```
Welcome Sachin Yadav to the world of functions in python.
```

Without Argument Without Return Type

[91]:
```
def greet():
    print("Very Very Welcome to Function in Python.")
    print("Functions are use to reuse statements.")
```

[92]:
```
help(greet)
```

```
Help on function greet in module __main__:

greet()
```

[93]:
```
print(greet.__doc__)
```

```
None
```

[94]:
```
greet()
```

```
Very Very Welcome to Function in Python.
Functions are use to reuse statements.
```

[95]:
```
value = greet()
print(value)
```

```
Very Very Welcome to Function in Python.
Functions are use to reuse statements.
None
```

[96]:
```
for var in range(3):
    greet()
    print('\n', "_"*60, '\n')
```

```
Very Very Welcome to Function in Python.
Functions are use to reuse statements.


 ------------------------------------------------------------

Very Very Welcome to Function in Python.
Functions are use to reuse statements.


 ------------------------------------------------------------

Very Very Welcome to Function in Python.
Functions are use to reuse statements.
```

With Argument Without Return Type

[101]:
```
def test_drink(name, age): # formal parameters
    """
        here name, age are the property of test_drink function

        We call them local variables or local parameters or local arguments

        these are famously known as keyword arguments
    """
    if age >= 18:
        print(f"{name.title()} can drink and can drive.")
    else:
        print(f"No!!!! {name.title()} beta, you are a kid you can not drink␣
↪neither you can drive.")
```

[105]:
```
value = test_drink('Sachin yadav', 24) # actual arguments
print(value)
```

```
Sachin Yadav can drink and can drive.
None
```

[103]:
```
test_drink('Rohit', 13)
```

```
No!!!! Rohit beta, you are a kid you can not drink neither you can drive.
```

[104]:
```
test_drink('ajay', 90)
```

```
Ajay can drink and can drive.
```

Without Argument With Return Type

[107]:
```
def squre():
    """
        squre() --> takes two number input and return square sum of their
    """
    x  = int(input("Enter x: "))
    y = int(input("Enter y: "))
    return x**2 + y**2
```

[108]:
```
ans = squre()
print("Answere: ", ans)
```

```
Enter x: 5
Enter y: 15
Answere:  250
```

With Argument With Return Type

[109]:
```
def is_even(number):
    """
        even_odd(number) --> True if number is even else False
    """
    if number % 2 == 0:
        return True
    else:
        return False
```

[110]:
```
is_even(10)
```

[110]:
```
True
```

[111]:
```
is_even(13)
```

[111]:
```
False
```

[112]:
```
def table(num):
    """
        table(num) --> print out table of a num
    """
    for var in range(1, 11):
        print(f"{num:>5} x {var:>2} = {var*num:>5}")
```

[113]:
```
table(5)
```

```
    5 x  1 =     5
    5 x  2 =    10
    5 x  3 =    15
    5 x  4 =    20
    5 x  5 =    25
    5 x  6 =    30
    5 x  7 =    35
    5 x  8 =    40
    5 x  9 =    45
    5 x 10 =    50
```

[114]:
```
table(3)
```

```
    3 x  1 =     3
    3 x  2 =     6
    3 x  3 =     9
    3 x  4 =    12
    3 x  5 =    15
    3 x  6 =    18
    3 x  7 =    21
```

```
3 x  8 =    24
3 x  9 =    27
3 x 10 =    30
```

[115]:
```
for num in range(5, 11):
    print()
    print("_"*60)
    print(f"\n\t\t\t\tTable of {num}\n\n")
    table(num)
    print("\n\n")
    print('_'*60)
```

```
------------------------------------------------------------

                                Table of 5


    5 x  1 =     5
    5 x  2=    10
    5 x  3 =    15
    5 x  4 =    20
    5 x  5 =    25
    5 x  6 =    30
    5 x  7 =    35
    5 x  8 =    40
    5 x  9 =    45
    5 x 10 =    50



------------------------------------------------------------

------------------------------------------------------------

                                Table of 6


    6 x  1 =     6
    6 x  2 =    12
    6 x  3 =    18
    6 x  4 =    24
    6 x  5 =    30
    6 x  6 =    36
    6 x  7 =    42
    6 x  8 =    48
    6 x  9 =    54
    6 x 10 =    60
```

Table of 7

```
7 x  1 =     7
7 x  2 =    14
7 x  3 =    21
7 x  4 =    28
7 x  5 =    35
7 x  6 =    42
7 x  7 =    49
7 x  8 =    56
7 x  9 =    63
7 x 10 =    70
```

Table of 8

```
8 x  1 =     8
8 x  2 =    16
8 x  3 =    24
8 x  4 =    32
8 x  5 =    40
8 x  6 =    48
8 x  7 =    56
8 x  8 =    64
8 x  9 =    72
8 x 10 =    80
```

Table of 9

```
 9 x  1 =     9
 9 x  2 =    18
 9 x  3 =    27
 9 x  4 =    36
 9 x  5 =    45
 9 x  6 =    54
 9 x  7 =    63
 9 x  8 =    72
 9 x  9 =    81
 9 x 10 =    90
```

```
                          Table of 10
```

```
10 x  1 =    10
10 x  2 =    20
10 x  3 =    30
10 x  4 =    40
10 x  5 =    50
10 x  6 =    60
10 x  7 =    70
10 x  8 =    80
10 x  9 =    90
10 x 10 =   100
```

---

prime

armstrong

fab(n)

pattern

tic-toc-toe

- Types of Arguments
- Scope in Function & Recursion
- Advance Python Function
- Decorators and Generators

- OOPs

[ ]: