02.2_Functions

August 30, 2023

1 Introduction to Python for Open Source Geocomputation

- Instructor: Dr. Wei Kang

Content:

- Functions

2 Functions - What is a function?

Functions are ways we can extend Python by writing code to add functionality that we would like to **reuse**.

- built-in functions: `print`, `help`
- functions written by other developers and published in a package: `numpy.log`
- user-defined functions

[1]:
```
print(10)
```

```
10
```

[2]:
```
help(print)
```

```
Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.
```

```
    sep
      string inserted between values, default a space.
    end
      string appended after the last value, default a newline.
    file
      a file-like object (stream); defaults to the current sys.stdout.
    flush
      whether to forcibly flush the stream.
```

[3]:
```
import numpy
```

Defined variables in the numpy package:

[4]:
```
numpy.inf
```

[4]:
```
inf
```

[5]:
```
numpy.inf > 1000
```

[5]:
```
True
```

Defined functions in the numpy package:

[6]:
```
help(numpy.log)
```

```
Help on ufunc:

log = <ufunc 'log'>
    log(x, /, out=None, *, where=True, casting='same_kind', order='K',
dtype=None, subok=True[, signature, extobj])

    Natural logarithm, element-wise.

    The natural logarithm `log` is the inverse of the exponential function,
    so that `log(exp(x)) = x`. The natural logarithm is logarithm in base
    `e`.

    Parameters
    ----------
    x : array_like
        Input value.
    out : ndarray, None, or tuple of ndarray and None, optional
        A location into which the result is stored. If provided, it must have
        a shape that the inputs broadcast to. If not provided or None,
        a freshly-allocated array is returned. A tuple (possible only as a
        keyword argument) must have length equal to the number of outputs.
    where : array_like, optional
```

```
    This condition is broadcast over the input. At locations where the
    condition is True, the `out` array will be set to the ufunc result.
    Elsewhere, the `out` array will retain its original value.
    Note that if an uninitialized `out` array is created via the default
    ``out=None``, locations within it where the condition is False will
    remain uninitialized.
**kwargs
    For other keyword-only arguments, see the
    :ref:`ufunc docs <ufuncs.kwargs>`.

Returns
-------
y : ndarray
    The natural logarithm of `x`, element-wise.
    This is a scalar if `x` is a scalar.

See Also
--------
log10, log2, log1p, emath.log

Notes
-----
Logarithm is a multivalued function: for each `x` there is an infinite
number of `z` such that `exp(z) = x`. The convention is to return the
`z` whose imaginary part lies in `(-pi, pi]`.

For real-valued input data types, `log` always returns real output. For
each value that cannot be expressed as a real number or infinity, it
yields ``nan`` and sets the `invalid` floating point error flag.

For complex-valued input, `log` is a complex analytical function that
has a branch cut `[-inf, 0]` and is continuous from above on it. `log`
handles the floating-point negative zero as an infinitesimal negative
number, conforming to the C99 standard.

In the cases where the input has a negative real part and a very small
negative complex part (approaching 0), the result is so close to `-pi`
that it evaluates to exactly `-pi`.

References
----------
.. [1] M. Abramowitz and I.A. Stegun, "Handbook of Mathematical Functions",
       10th printing, 1964, pp. 67.
       https://personal.math.ubc.ca/~cbm/aands/page_67.htm
.. [2] Wikipedia, "Logarithm". https://en.wikipedia.org/wiki/Logarithm

Examples
--------
```

```
>>> np.log([1, np.e, np.e**2, 0])
array([  0.,   1.,   2., -Inf])
```

[7]:
```
numpy.log(10)
```

[7]:
```
2.302585092994046
```

2.1 Python Keywords

- predefined and reserved words in python that have special meanings
- used to define the syntax of the coding
- cannot be used as an identifier, function, or variable name
- All the keywords in python are written in lowercase except `True` and `False`.

2.2 Components of a user-defined function

- Required:
  - function keyword `def`
  - function name and parentheses
  - solution: statements/expressions
- Optional
  - parameters(s) (input)
    - A parameter is the variable listed inside the parentheses in the function definition.
  - `return` Statement (output)
    - `return` followed by an optional return value
    - immediately terminates a function execution and sends the return value back to the caller code

2.2.1 An example of a function

```
def square(x):
    return x*x
```

- `def`: Python keyword for function definition
- `square`: function name
- `x`: parameter
- `return`: what comes out of the function

[8]:
```
def square(x):
    return x*x
```

[9]:
```
square(10)
```

[9]:
```
100
```

[10]:
```
type(square)
```

[10]:
```
function
```

2.2.2 Function Calls

Call the function `square` to calculate the square of 10?

- function name
- paratheses
- argument(s): input
  - An argument is the value that is sent to the function when it is called.
  - Should match parameters in the function definition

[11]:
```
square(10)
```

[11]:
```
100
```

[12]:
```
type(square)
```

[12]:
```
function
```

The function `square` takes the argument of 10 and return the value of 100

We can also assign the returned value to a new variable

[13]:
```
squared_10 = square(10)
```

[14]:
```
squared_10
```

[14]:
```
100
```

[15]:
```
type(squared_10)
```

[15]:
```
int
```

[16]:
```
type(square(10))
```

[16]:
```
int
```

[17]:
```
type(square)
```

[17]:
```
function
```

2.2.3 Difference between `print` and `return` inside a function

- `return`: sends the calculated value back to the caller code
  - `return` is a keyword
  - `return 1`
- `print`: display formatted messages onto the screen
  - `print` is a function
  - `print(1)`

[18]:
```
def square(x):
    return x*x
```

[19]:
```
x_squared = square(10)
```

[20]:
```
x_squared
```

[20]: 100

[21]:
```
def square(x):
    print(x*x)
```

[22]:
```
x_squared = square(10)
```

```
100
```

[23]:
```
x_squared
```

[24]:
```
type(x_squared)
```

[24]: NoneType

2.2.4 Composition

We can also use the function `square` on the returned value of `square`

[25]:
```
def square(x):
    return x*x
```

[26]:
```
square(square(10))
```

[26]: 10000

[27]:
```
squared_10 = square(10)
square(squared_10)
```

[27]: 10000

[28]:
```
square_10 = 10
for i in range(3):
    square_10 = square(square_10)
square_10
```

[28]: 100000000

2.3 Void functions: no Return values

```
def hello():
    print("Hello World!")
```

- What are the components of the function?
- What components does this function have?
- What components does this function not have?

[29]:
```
def hello():
    print("Hello World!")
```

[30]:
```
hello()
```

```
Hello World!
```

[31]:
```
b = hello()
```

```
Hello World!
```

[32]:
```
b
```

[33]:
```
type(b)
```

[33]:
```
NoneType
```

[34]:
```
def hello():
    return "luck!"
```

[35]:
```
b = hello()
```

[36]:
```
b
```

[36]:
```
'luck!'
```

[37]:
```
type(b)
```

[37]:
```
str
```

[38]:
```
def hello():
    return "luck!"
    print("luck!")
```

[39]:
```
def hello():
    print("luck!")
    return "luck!"
```

- `def`: Python keyword for function definition
- `hello`: function name

- `print("Hello World!")`: print statement

2.4 Functions with more than 1 parameters

```
def addition(x, y):
    return x+y
```

[40]:
```
def addition(x, y):
    return x+y
```

[41]:
```
addition
```

[41]:
```
<function __main__.addition(x, y)>
```

[42]:
```
addition(1,2)
```

[42]:
```
3
```

[43]:
```
1+2
```

[43]:
```
3
```

[44]:
```
def deduction(x,y):
    return x-y
```

[45]:
```
deduction(2,1)
```

[45]:
```
1
```

[46]:
```
deduction(1,2)
```

[46]:
```
-1
```

[47]:
```
def deduction_reverse(x,y):
    return y-x
```

[48]:
```
deduction_reverse(2,1)
```

[48]:
```
-1
```

[49]:
```
x
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[49], line 1
----> 1 x
```

```
NameError: name 'x' is not defined
```

[ ]:
```
x = 10
```

[ ]:
```
x
```

[ ]:
```
def deduction_reverse(x,y):
    x = 100
    return y-x
```

[ ]:
```
deduction_reverse(x,1)
```

[ ]:
```
x
```

2.4.1 Parameters/Arguments in a Function

- order/position is important
- local variables - cannot be used outside the function

2.4.2 Group Exercise:

Suppose the cover price of a book is \$24.95, but bookstores get a 40% discount. Shipping costs \$3 for the first copy and 75 cents for each additional copy. What is the total wholesale cost for 60 copies? 15 copies? 10000 copies?

> how to automate our solution in a function? What are the input (arguments)? How should we select the arguments?

```
def total_costs(num_copies):
    return
```

[ ]:
```
def total_costs(num_copies):
    return (24.95 * (1-0.4) * num_copies) + (3 *1 + (num_copies-1) * 0.75)
```

[ ]:
```
num_copies = 88
```

[ ]:
```
total_costs(num_copies)
```

[ ]:
```
total_costs(99)
```

[ ]:
```
copies = 999
```

[ ]:
```
total_costs(copies)
```

[ ]:
```
total_costs(999)
```

[ ]:
```
def total_costs(num_copies, discount):
#     num_copies = 60
    return (24.95 * (1-discount) * num_copies) + (3 *1 + (num_copies-1) * 0.75)
```

[ ]:
```
total_costs(99, 0.1)
```

2.5 Good programming practice with functions - docstrings

Numpy docstring guidlines https://numpydoc.readthedocs.io/en/latest/format.html

- A one-line summary describing what the function does
- Description of the function arguments and their respective types. (input)
- Description of the function's returns and their respective types. (output)
- Some examples (optional)

[ ]:
```
numpy.log?
```

2.6 Abundant functions in Python

- built-in function:
  - `type`: get an object's type
  - `int`: Convert a number or string to an integer
- functions from built-in modules:
  - import statement: `import math` creates a module object named math
  - The `math` module object contains the functions and variables defined in the module
  - how to access these functions or variables: dot notation (math.log)

[ ]:
```
type(10)
```

[ ]:
```
type(10.1)
```

[ ]:
```
int(10.1)
```

[ ]:
```
int(10.9)
```

[ ]:
```
import math
dir(math)
```

2.6.1 Variables and functions in the imported module

[ ]:
```
math.pi
```

2.6.2 Group Exercise

- calculate the smallest integer that is larger than π
- calculate the largest integer that is smaller than π

Hint: use the `int` function

[ ]:
```
int(math.pi)+1
```

[ ]:
```
int(math.pi)
```

[ ]:
```
math.ceil?
```

[ ]:
```
math.ceil(math.pi)
```

3 HW1 (Programming Assignment)

- Where: You will use your UNT EUID and password to login the Jupyter Hub https://jupyterhub.cas.unt.edu/ to complete the assignment and submit it. You need to make sure to connect to UNT VPN beforehand.
- Four programming exercises
  - complete the solution in a function
  - the function is defined
  - Use the test case to evaluate whether your solution is correct
  - submit your completed work to the Jupyter Hub by 09/07/2023
    - Do not submit multiple times as I can only see your last submission and submission time (risk of being considered as late work)
    - 24-hour extension, no tokens needed
    - Start early: getting an addtional token by handing in an assignment at least 24 hours before its due date.

4 Next Class

Topics:

- Numerical data types

Readings: Chapter 5

[ ]:
```

```