03.1_Functions

February 6, 2023

1 Introduction to Python for Open Source Geocomputation

- Instructor: Dr. Wei Kang
- Class Location and Time: ENV 336, Mon & Wed 12:30 pm - 1:50 pm

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]:
```
help(print)
```

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

print(…)
    print(value, …, sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
```

```
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

[2]:
```
import numpy
```

Defined variables in the numpy package:

[3]:
```
numpy.inf
```

[3]:
```
inf
```

[4]:
```
numpy.inf > 1000
```

[4]:
```
True
```

Defined functions in the numpy package:

[5]:
```
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.

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])
```

[6]:
```
numpy.log(10)
```

[6]:
```
2.302585092994046
```

2.1 Components of a user-defined function

- Required:
  - function keyword *def*
  - function name
  - solution: statements/expressions
- Optional
  - Argument(s) (input)
  - Return values (output)

[7]:
```
def square(x):
    return x*x
```

2.1.1 An example of function

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

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

[ ]:
```

```

[8]:
```
def function_name():
    print("happy!")
```

[9]:
```
function_name()
```

```
happy!
```

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

[10]:
```
int
```

[11]:
```
type("hello")
```

[11]:
```
str
```

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

[12]:
```
function
```

[13]:
```
type(function_name())
```

```
happy!
```

[13]:
```
NoneType
```

[14]:
```
def square(x):
    return x*x
```

2.1.2 Function Calls

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

[15]:
```
square(10)
```

[15]: 100

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

[16]: function

[17]:
```
type(square(10))
```

[17]: int

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

[18]:
```
squared_10 = square(10)
```

[19]:
```
squared_10
```

[19]: 100

2.1.3 Composition

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

[20]:
```
square(square(10))
```

[20]: 10000

[21]:
```
squared_10 = square(10)
square(squared_10)
```

[21]: 10000

2.2 Void functions: no Return values

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

What are the components of the function?

[22]:
```
def hello():
    return "luck!"
    print("Hello World!")
```

[23]:
```
b = hello()
```

[24]:
```
b
```

[24]: 'luck!'

[25]:
```
def hello():
    print("Hello World!")
    return "luck!"
```

[26]:
```
b = hello()
```

```
Hello World!
```

[27]:
```
b == "luck!"
```

[27]: True

[28]:
```
b
```

[28]: 'luck!'

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

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

```
Hello World!
```

[31]:
```
b
```

[32]:
```
type(b)
```

[32]: NoneType

[33]:
```
b == "luck"
```

[33]: False

[34]:
```
assert b == "luck"
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
```

```
/var/folders/6m/8n2ktxl566j8yp0n_qx7x5bw0000gt/T/ipykernel_63243/3769617147.py␣
 ↪in <module>
----> 1 assert b == "luck"

AssertionError:
```

- `def`: Python keyword for function definition
- `hello`: function name
- `print("Hello World!")`: print statement

2.3 Functions with more than 1 arguments

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

[35]:
```
def addition(x, y):
    return x+y
```

[36]:
```
addition(1,2)
```

[36]: 3

[37]:
```
1+2
```

[37]: 3

[38]:
```
def deduction(x,y):
    return x-y
```

[39]:
```
deduction(2,1)
```

[39]: 1

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

[40]: -1

[41]:
```
def deduction_reverse(x,y):
    return y-x
```

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

[42]: -1

[43]:
```
x
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
```

```
/var/folders/6m/8n2ktxl566j8yp0n_qx7x5bw0000gt/T/ipykernel_63243/32546335.py in
 ↪<module>
----> 1 x

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

[44]:
```
x = 10
```

[45]:
```
x
```

[45]: 10

[46]:
```
def deduction_reverse(x,y):
    x = 100
    return y-x
```

[47]:
```
deduction_reverse(x,1)
```

[47]: -99

[48]:
```
x
```

[48]: 10

[49]:
```
deduction(1,2)
```

[49]: -1

[50]:
```
deduction(2,1)
```

[50]: 1

2.3.1 Arguments in a Function

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

2.3.2 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?

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

[51]:
```
def total_costs(num_copies):
    return (24.95 * (1-0.4) * num_copies) + (3 *1 + (num_copies-1) * 0.75)
```

[52]:
```
total_costs(99)
```

[52]:
```
1558.53
```

[53]:
```
def total_costs(num_copies, discount):
#     num_copies = 60
    return (24.95 * (1-discount) * num_copies) + (3 *1 + (num_copies-1) * 0.75)
```

[54]:
```
total_costs(99, 0.1)
```

[54]:
```
2299.5449999999996
```

2.3.3 Exercise:

Suppose the cover price of a book is \$56, 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 15 copies?

When you are done with your calculation, raise your hands!

2.4 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)

2.5 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)

[55]:
```
type(10)
```

[55]:
```
int
```

[56]:
```
type(10.1)
```

[56]:
```
float
```

[57]:
```
int(10.1)
```

[57]: 10

[58]:
```
int(10.9)
```

[58]: 10

[59]:
```
import math
dir(math)
```

[59]:
```
['__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'acos',
 'acosh',
 'asin',
 'asinh',
 'atan',
 'atan2',
 'atanh',
 'ceil',
 'comb',
 'copysign',
 'cos',
 'cosh',
 'degrees',
 'dist',
 'e',
 'erf',
 'erfc',
 'exp',
 'expm1',
 'fabs',
 'factorial',
 'floor',
 'fmod',
 'frexp',
 'fsum',
 'gamma',
 'gcd',
 'hypot',
 'inf',
 'isclose',
 'isfinite',
 'isinf',
```

```
 'isnan',
 'isqrt',
 'lcm',
 'ldexp',
 'lgamma',
 'log',
 'log10',
 'log1p',
 'log2',
 'modf',
 'nan',
 'nextafter',
 'perm',
 'pi',
 'pow',
 'prod',
 'radians',
 'remainder',
 'sin',
 'sinh',
 'sqrt',
 'tan',
 'tanh',
 'tau',
 'trunc',
 'ulp']
```

2.5.1 Variables and functions in the imported module

[60]:
```
math.pi
```

[60]:
```
3.141592653589793
```

2.5.2 Exercise

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

Hint: use the `int` function

[61]:
```
int(math.pi)+1
```

[61]:
```
4
```

[62]:
```
int(math.pi)
```

[62]:
```
3
```

[63]:
```
math.ceil(math.pi)
```

[63]: 4

[64]:
```
math.floor(math.pi)
```

[64]: 3

3 Homework 1 (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 in the Jupyter Hub by 10 pm on 02/12/2023

4 Next Class

Topics:

- Numerical data types

Readings: Chapter 5

[ ]:
```

```