ERTH 401 / GEOP 501
Computational Methods for Geoscientists

– Lecture 04: Functions –

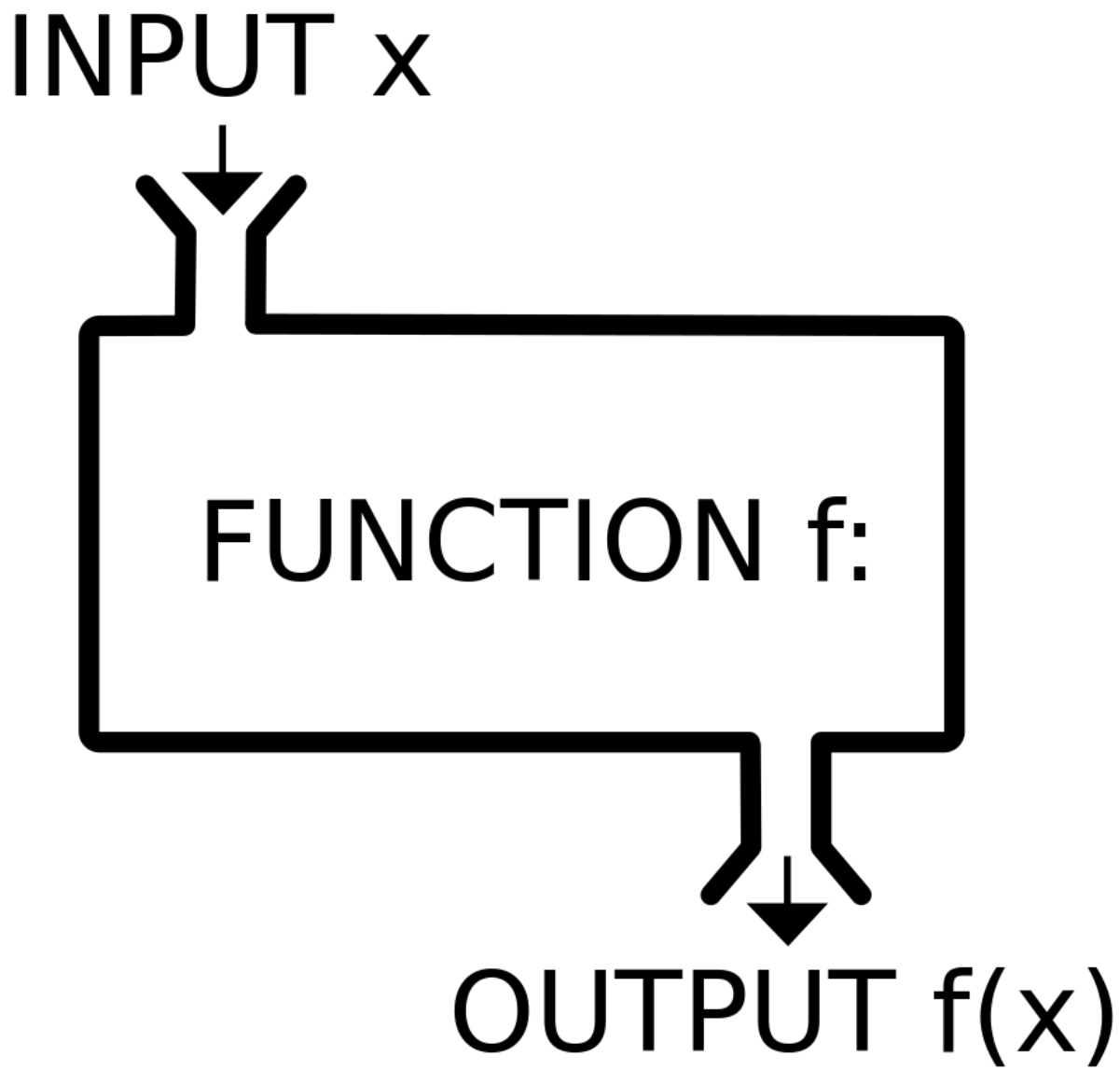
Ronni Grapenthin
rg@nmt.edu
MSEC 356
x5924

September 18, 2017

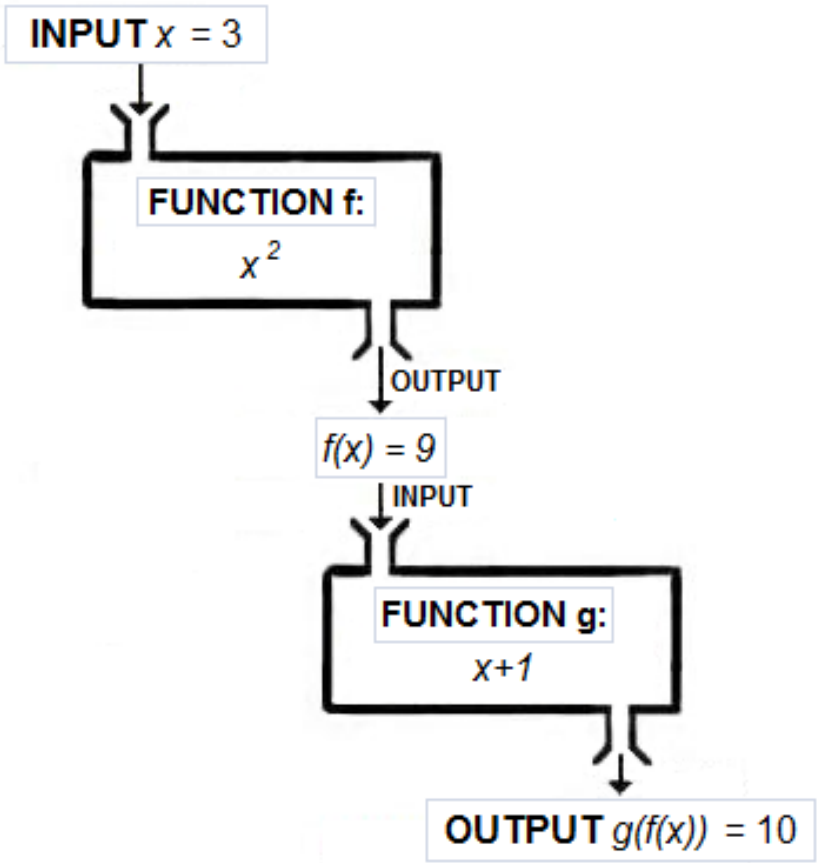
Wouldn’t it be great if we could think up some functionality, write it up, test it, and be able to reuse it at any time in the future?*

* hint: yes!

Function

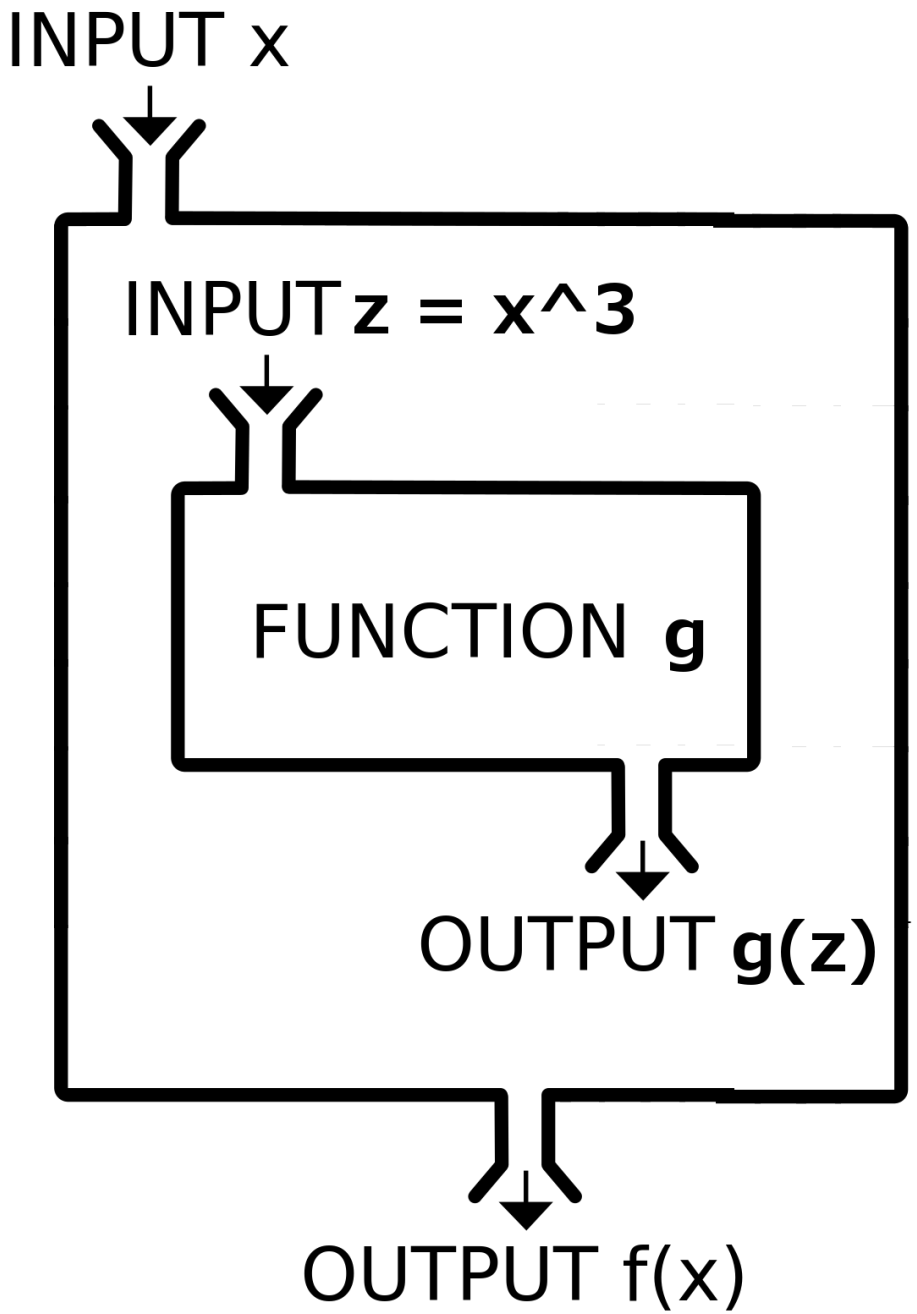


Function & Function



https://en.wikipedia.org

Use a Function in a Function



What Is A Function?

. . . a set of rules that takes input, from which it produces output.

What Is A Function?

- one of the boxes you drew in the flow charts
- one of the most important tools you have to:
  - capture & contain
  - test
  - reuse

  algorithms / behavior of any complexity
- change something in **one** spot instead of all over your source code (don't copy and paste in programming!)

Examples

```
1  #!/usr/bin/env python

3  def letter_grade(percent):
       if   percent > 90:
5          return "A"
       elif percent > 80:
7          return "B"
       elif percent > 70:
9          return "C"
       elif percent > 60:
11         return "D"
       else:
13         return "F"

15 if __name__ == "__main__":
       print letter_grade(85)
```

```
$> ./grades.py
B
```

Examples

```python
#!/usr/bin/env python

def letter_grade(percent):
    if   percent > 90:
        return "A"
    elif percent > 80:
        return "B"
    elif percent > 70:
        return "C"
    elif percent > 60:
        return "D"
    else:
        return "F"

if __name__ == "__main__":
    print letter_grade(85)
```

```bash
#!/bin/bash
function letter_grade(){
  if   [ $1 -gt 90 ]; then
       echo "A"
  elif [ $1 -gt 80 ]; then
       echo "B"
  elif [ $1 -gt 70 ]; then
       echo "C"
  elif [ $1 -gt 60 ]; then
       echo "D"
  else
       echo "F"
  fi
}

letter_grade 85
```

```
$> ./grades.py
B
```

```
$>./letter_grades.bash
B
```

Examples - Interactive Python

import **entire module**

```
>>> import grades
>>> grades.letter_grade(60)
'F'
>>> grades.letter_grade(87)
'B'
>>> g = grades.letter_grade(87)
>>> print g
B
```

Examples - Interactive Python

import **entire module**

```
>>> import grades
>>> grades.letter_grade(60)
'F'
>>> grades.letter_grade(87)
'B'
>>> g = grades.letter_grade(87)
>>> print g
B
```

import **individual function** from a module

```
>>> from grades import letter_grade
>>> letter_grade(60)
'F'
>>> letter_grade(87)
'B'
>>> g = letter_grade(87)
>>> print g
B
```

Examples - bash vs tcsh

```
#!/bin/bash
function letter_grade(){
  if [ $1 -gt 90 ]; then
        echo "A"
  elif [ $1 -gt 80 ]; then
        echo "B"
  elif [ $1 -gt 70 ]; then
        echo "C"
  elif [ $1 -gt 60 ]; then
        echo "D"
  else
        echo "F"
  fi
}

letter_grade $1
```

```
$> ./letter_grades2.bash 85
B
```

Examples - bash vs tcsh

```bash
#!/bin/bash
function letter_grade(){
    if [ $1 -gt 90 ]; then
        echo "A"
    elif [ $1 -gt 80 ]; then
        echo "B"
    elif [ $1 -gt 70 ]; then
        echo "C"
    elif [ $1 -gt 60 ]; then
        echo "D"
    else
        echo "F"
    fi
}

letter_grade $1
```

```tcsh
#!/bin/tcsh

set grade = $1


if    ($grade > 90) then
        echo "A"
else if ($grade > 80) then
        echo "B"
else if ($grade > 70) then
        echo "C"
else if ($grade > 60) then
        echo "D"
else
        echo "F"
endif
```

```
$> ./letter_grades2.bash 85
B
```

```
$> ./letter_grades.tcsh 85
B
```

Overview

- programming languages provide functions differently
- functions in the shell are a bit clunky (bash) or not available (tcsh), but workarounds possible
- think about how to parameterize your function

Syntax

- some languages provide multiple return values:

```
#!/usr/bin/env python

def get_two_values():
    return 23, 42

if __name__ == "__main__":
    val1, val2 = get_two_values()
    print val1
    print val2
```

```
$> ./multi_return.py
23
42
```

Syntax

- some languages provide multiple return values:

```
1 #!/usr/bin/env python

3 def get_two_values():
      return 23, 42
5
  if __name__ == "__main__":
7     val1, val2 = get_two_values()
      print val1
9     print val2
```

```
$> ./multi_return.py
23
42
```

- could always return more complex type with multiple values (arrays, lists, . . .)

Examples - Python: multiple functions

```
1  #!/usr/bin/env python

3  def letter_grade(percent):
       if   percent > 90:
5          return "A"
       elif percent > 80:
7          return "B"
       elif percent > 70:
9          return "C"
       elif percent > 60:
11         return "D"
       else:
13         return "F"

15 def gpa(letter_grades, credits):
       grades={'A': 4.00, 'B': 3.00, 'C':2.00, 'D':1.00, 'F':0.00}
17
       total_points = 0
19
       for i in range(len(letter_grades)):
21         total_points += grades[ letter_grades[i] ] * credits[i]

23     return float(total_points)/float(sum(credits))
```

Examples - Interactive Python

import **entire module**

```
>>> import grades2
>>> print grades2.gpa(['B','A','C','A','B'], [4, 3, 3, 4, 2])
3.25
>>> print grades2.letter_grade(90)
B
```

Examples - Interactive Python

import **entire module**

```
>>> import grades2
>>> print grades2.gpa(['B','A','C','A','B'], [4, 3, 3, 4, 2])
3.25
>>> print grades2.letter_grade(90)
B
```

import **individual functions** from a module

```
>>> from grades2 import gpa, letter_grade
>>> print gpa(['B','A','C','A','B'], [4, 3, 3, 4, 2])
3.25
>>> print letter_grade(90)
B
```

What is this {'A':4.0} Business?

- python provides `dictionary` datatype, associative arrays
- think "Lookup tables"
- you get to assign key value pairs where the key can be string, number, other datatype (list requires the key to be the index of the value)

What is this {'A':4.0} Business?

- python provides `dictionary` datatype, associative arrays
- think "Lookup tables"
- you get to assign key value pairs where the key can be string, number, other datatype (list requires the key to be the index of the value)

```
>>> grades={'A': 4.00, 'B': 3.00, 'C':2.00}
>>> grades['A']
4.0
>>> grades['a']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'a'
>>> grades['D'] = 2.00
>>> grades['F'] = 0.00
>>> grades['D']
2.0
>>> grades['D'] = 1.00
>>> grades['D']
1.0
>>> grades[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 4
```