Nested functions. Homework 2

Lecture 03.03

By *Marina Barsky*

Docstring: tests, type contracts, and preconditions

Following the Function Design Recipe, write a function that meets this description:

This function returns a string containing a particular word repeated a particular number of times.

For example, someone should be able to call the function to repeat "Marcia " three times and the function should return "Marcia Marcia Marcia ", or call the function to repeat "Buffalo " eight times and have it return "Buffalo Buffalo Buffalo Buffalo Buffalo Buffalo Buffalo Buffalo ".

1. Examples
2. Type Contract and preconditions
3. Header
4. Description
5. Code the Body
6. Test

Importing functions: 2 ways

```
import random
```

This is preferred – so you always know where the function is coming from

- To use:

```
random.choice (["rock", "paper", "scissors")
```

```
from random import *
```

- To use:

```
choice (["rock", "paper", "scissors")
```

Nested functions

functions that call other functions

How functions work: inside the machine

- When the program reaches the point of function call:
  - The calling program suspends execution at the point of the call
  - The parameters that are used by the function get assigned the values supplied for this call.
  - The body of the function is executed.
  - Control returns to the point just after where the function was called.

Functions can call other functions

```
def f (x):
    x = 2*x
    return x

def g (x):
    x = 2*f(x/3)
    return x

def h (x):
    x = 2*g(x/2)
    return x

#function call
h (6)
```

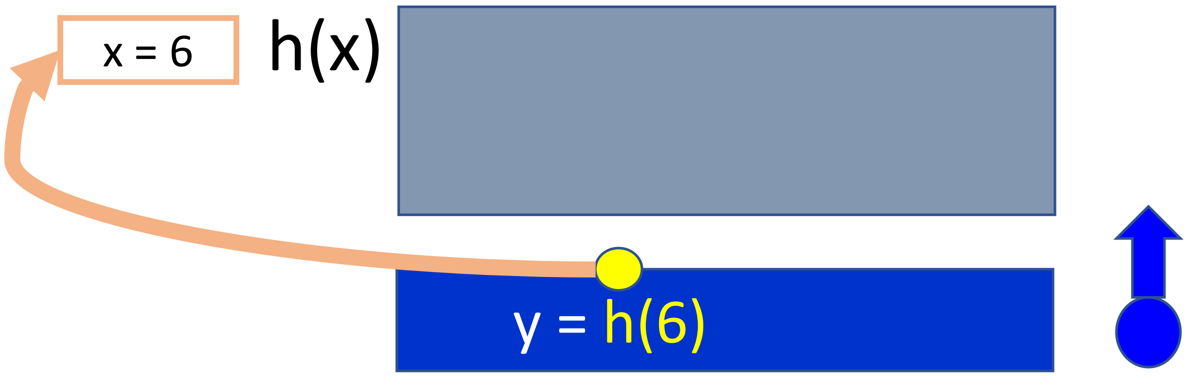
Stack – program is loaded, now calling h(x)

```
def f (x):
    x = 2*x
    return x

def g (x):
    x = 2*f(x/3)
    return x

def h (x):
    x = 2*g(x/2)
    return x

#function call
y  = h (6)
```



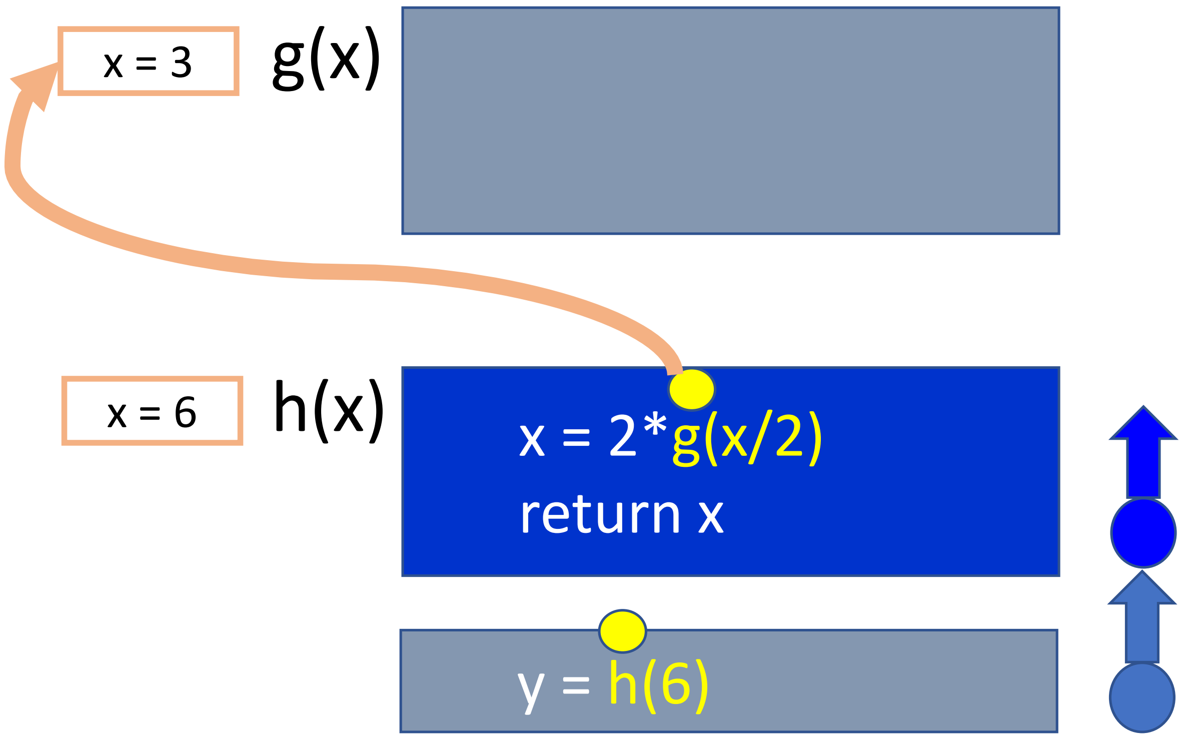
Stack – h(x) loaded, now calling g(x)

```
def f (x):
    x = 2*x
    return x

def g (x):
    x = 2*f(x/3)
    return x

def h (x):
    x = 2*g(x/2)
    return x

#function call
y  = h (6)
```



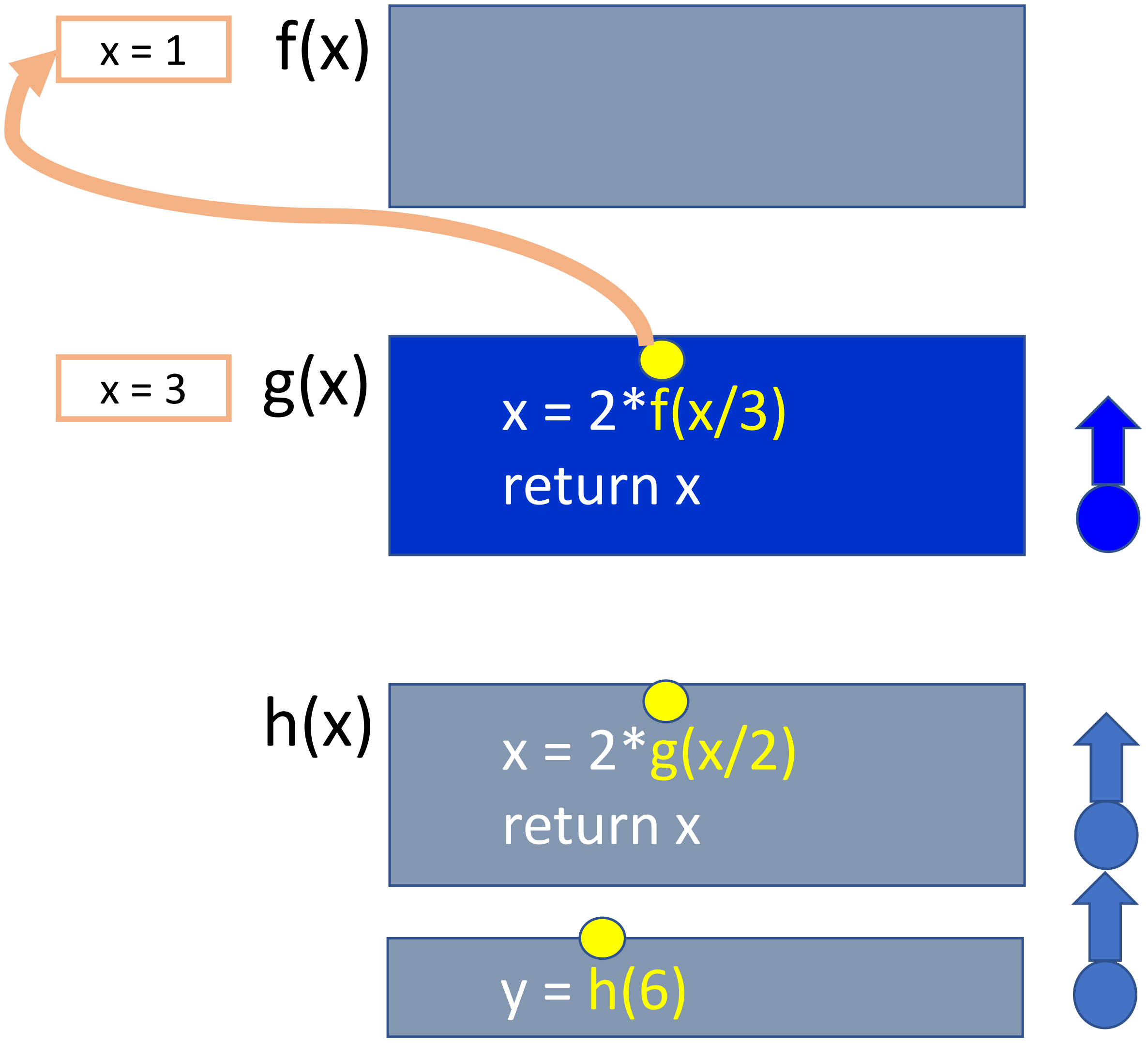
Stack – g(x) loaded, calling f(x)

```
def f (x):
    x = 2*x
    return x

def g (x):
    x = 2*f(x/3)
    return x

def h (x):
    x = 2*g(x/2)
    return x

#function call
y  = h (6)
```



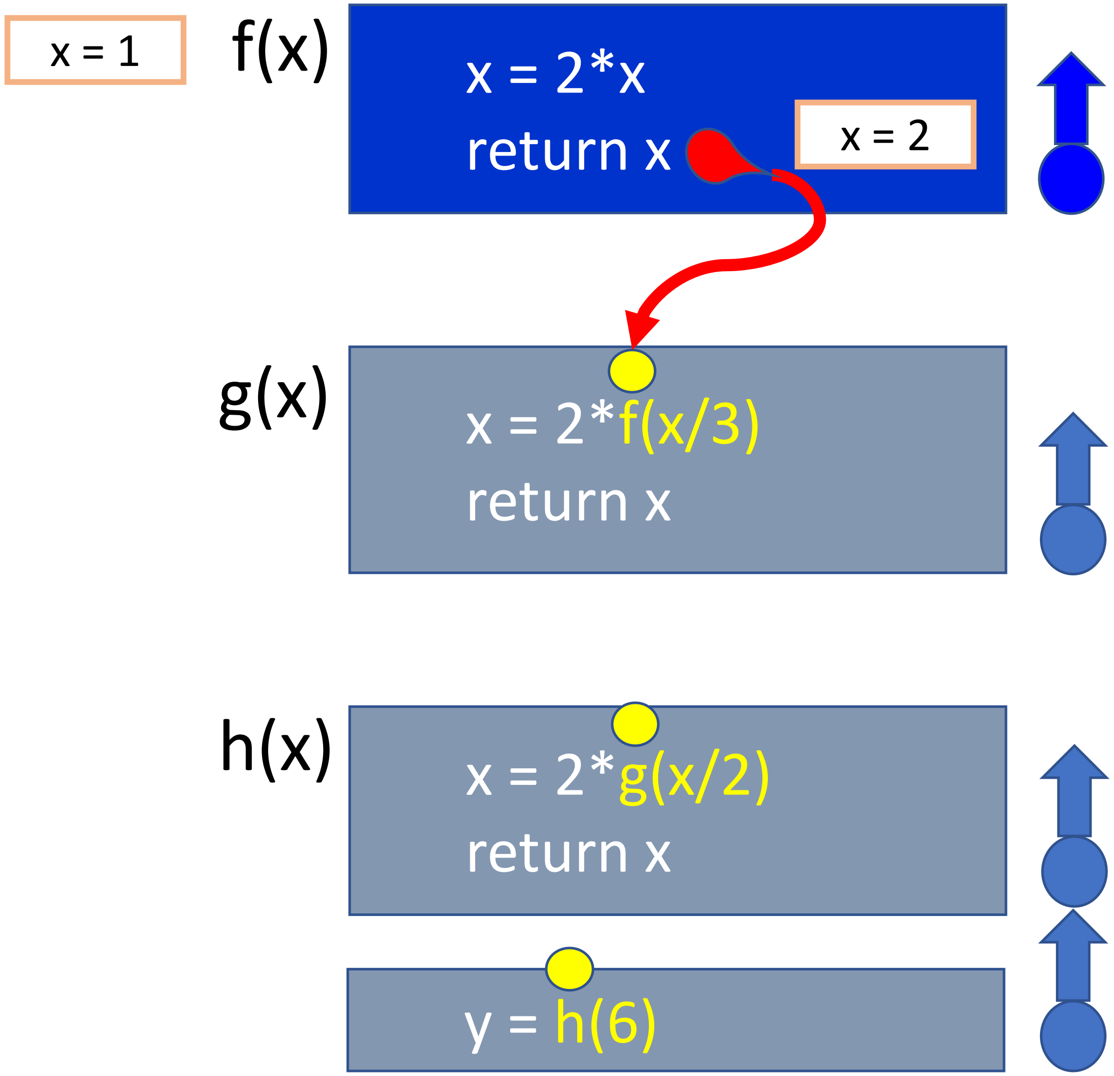
Stack – f(x) loaded, computing

```
def f (x):
    x = 2*x
    return x

def g (x):
    x = 2*f(x/3)
    return x

def h (x):
    x = 2*g(x/2)
    return x

#function call
y  = h (6)
```



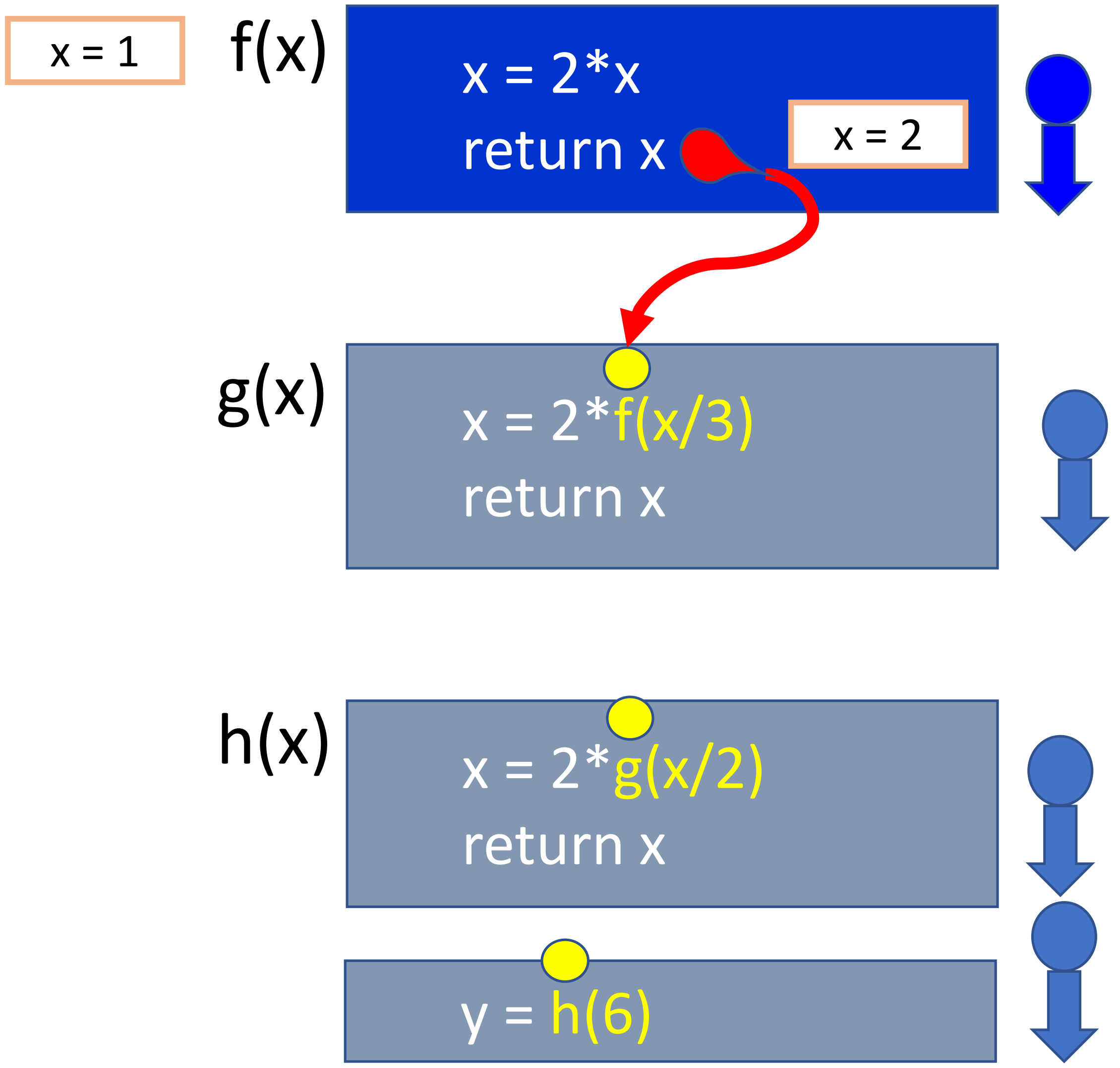
Stack – f(x) returning 2

```
def f (x):
    x = 2*x
    return x


def g (x):
    x = 2*f(x/3)
    return x


def h (x):
    x = 2*g(x/2)
    return x


#function call
y  = h (6)
```



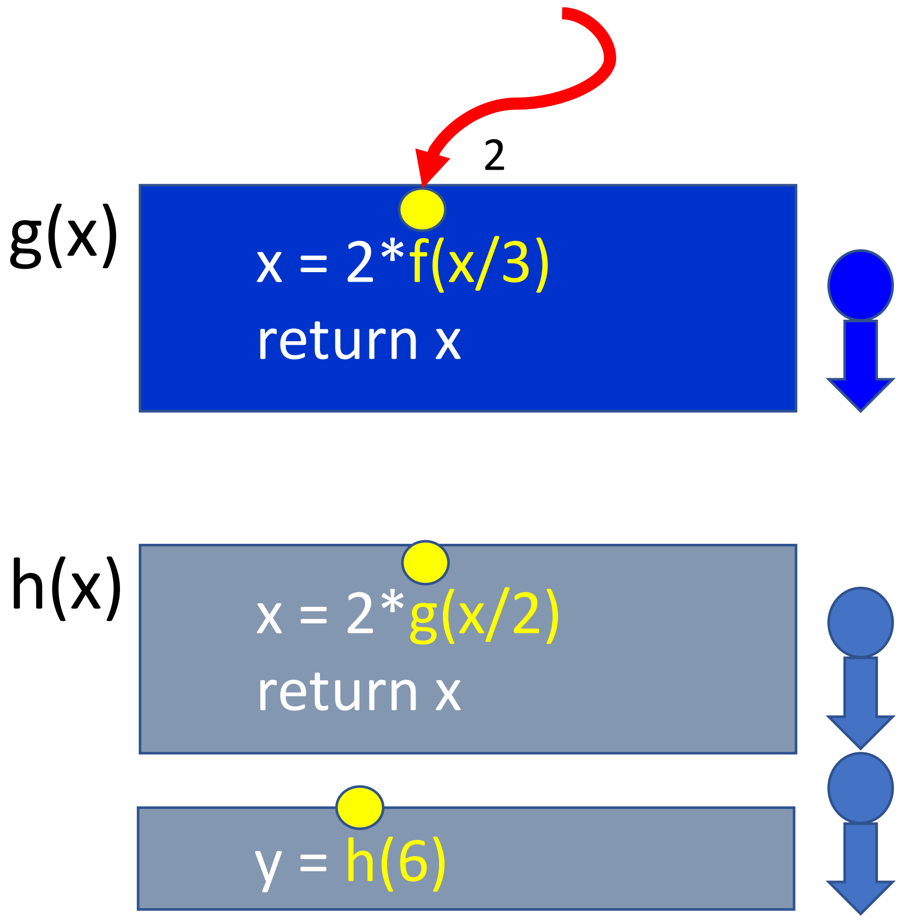
Stack – f(x) unloaded (exited), computing g(x)

```
def f (x):
    x = 2*x
    return x

def g (x):
    x = 2*f(x/3)
    return x

def h (x):
    x = 2*g(x/2)
    return x

#function call
y  = h (6)
```



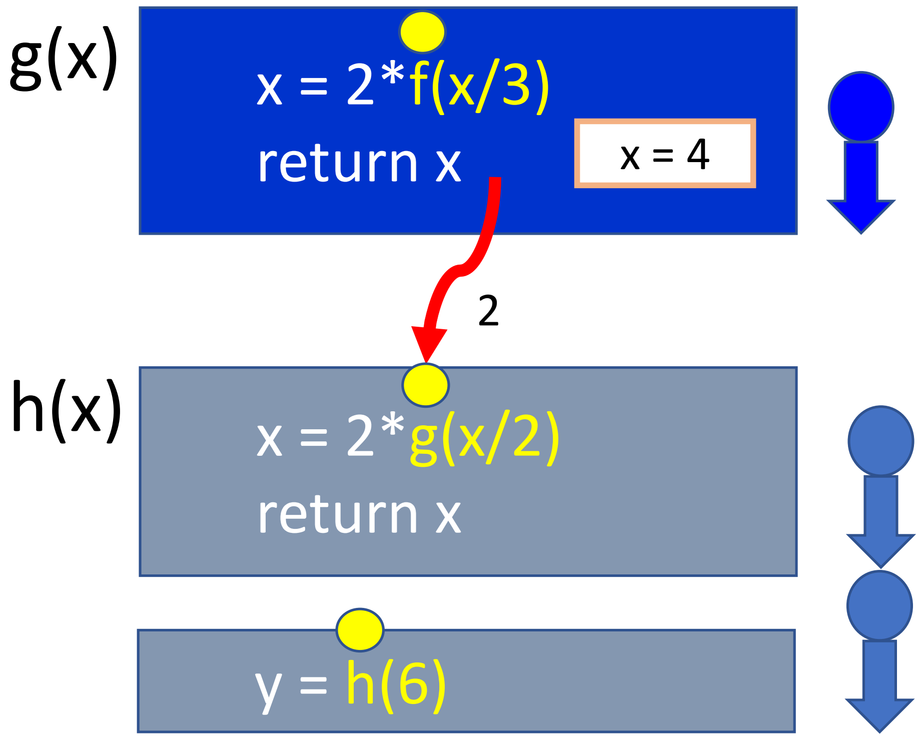
Stack – g(x) returning 4

```
def f (x):
    x = 2*x
    return x

def g (x):
    x = 2*f(x/3)
    return x

def h (x):
    x = 2*g(x/2)
    return x

#function call
y  = h (6)
```



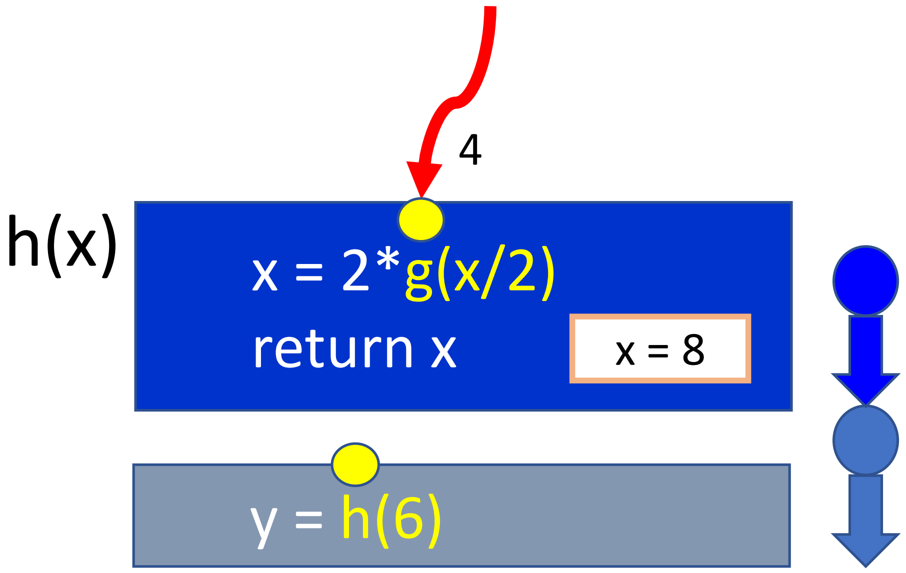
Stack – g(x) exited, computing h(x)

```
def f (x):
    x = 2*x
    return x


def g (x):
    x = 2*f(x/3)
    return x


def h (x):
    x = 2*g(x/2)
    return x


#function call
y  = h (6)
```



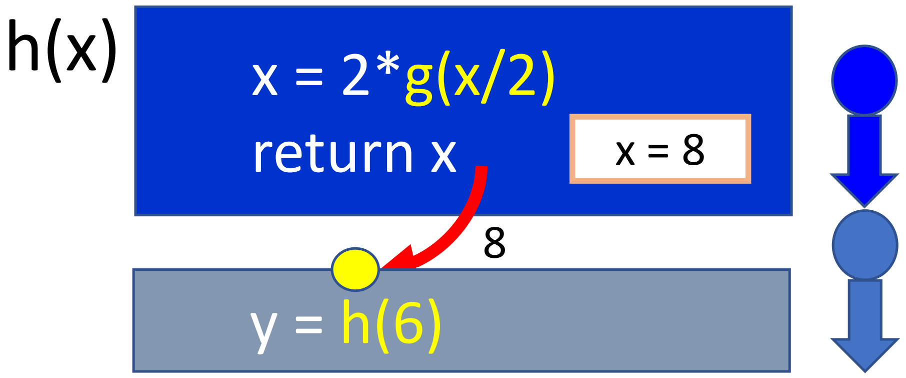
Stack – h(x) returns 8

```
def f (x):
    x = 2*x
    return x


def g (x):
    x = 2*f(x/3)
    return x


def h (x):
    x = 2*g(x/2)
    return x


#function call
y  = h (6)
```



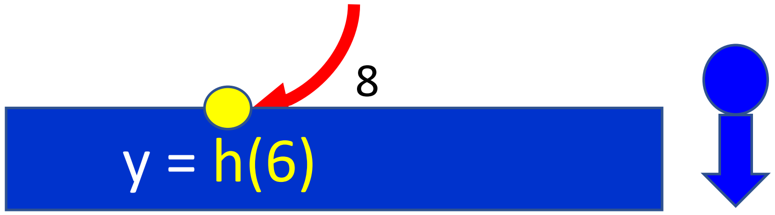
Stack – h(x) exited, placing 8 into variable y

```
def f (x):
    x = 2*x
    return x


def g (x):
    x = 2*f(x/3)
    return x


def h (x):
    x = 2*g(x/2)
    return x


#function call
y  = h (6)
```



Order of execution for nested functions

Function arguments are *evaluated* from left to right and only then the function itself is executed

```
f(g() + 3, h())
  ------        ------
    1             2
   -------------------
            3
                       ------
                         4
 ----------------------------------------
                  5
```

Example 1: max of min

- Underline expressions and number them in the order of evaluation

```
def max_of_min(num1, num2, value1, value2):
  return max(min(num1, num2), min(value1, value2))
                  ------    ------
                    1         2
```

Example 2: pow of pow

```
print(pow(2, pow(pow(2, 1), 2)))
          ------
           1
```

Use case:
function to produce roots of quadratic equations

- A quadratic equation is an equation that could be written as

$Ax^2 + Bx + C = 0$

- Our ultimate goal is to write a function solve(A,B,C), which will return the list of rational roots (if any) of the above equation.
- We are going to perform this task in steps, to exercise nested function design and function reuse.

Steps

1. Function num_roots(A, C) which returns the number of roots(0, 1, or 2) according to values of A,B,C
2. Function get_discriminant(A,B,C) which will compute the value of discriminant
3. Function get_roots(A, B, C) which will return a list of 1, 2, or 0 roots

Step 1

- num_roots(A,B, C)

- If $B^2 - 4AC$ is equal to 0 → return 1
- If $B^2 - 4AC$ is > 0 → 2
- Else → return 0

Step 2. Value of Discriminant: D = $\sqrt{B^2 - 4AC}$

- get_discriminant (A, B, C)

- If num_roots == 0 → return None
- If num_roots == 1 → return 0
- Else → return (B**2 – 4*A*C)**0.5

Step 3. List of roots: $x_1 = (-B + D)/(2*A)$, $x_2 = (-B - D)/(2*A)$

- solve(A, B, C)

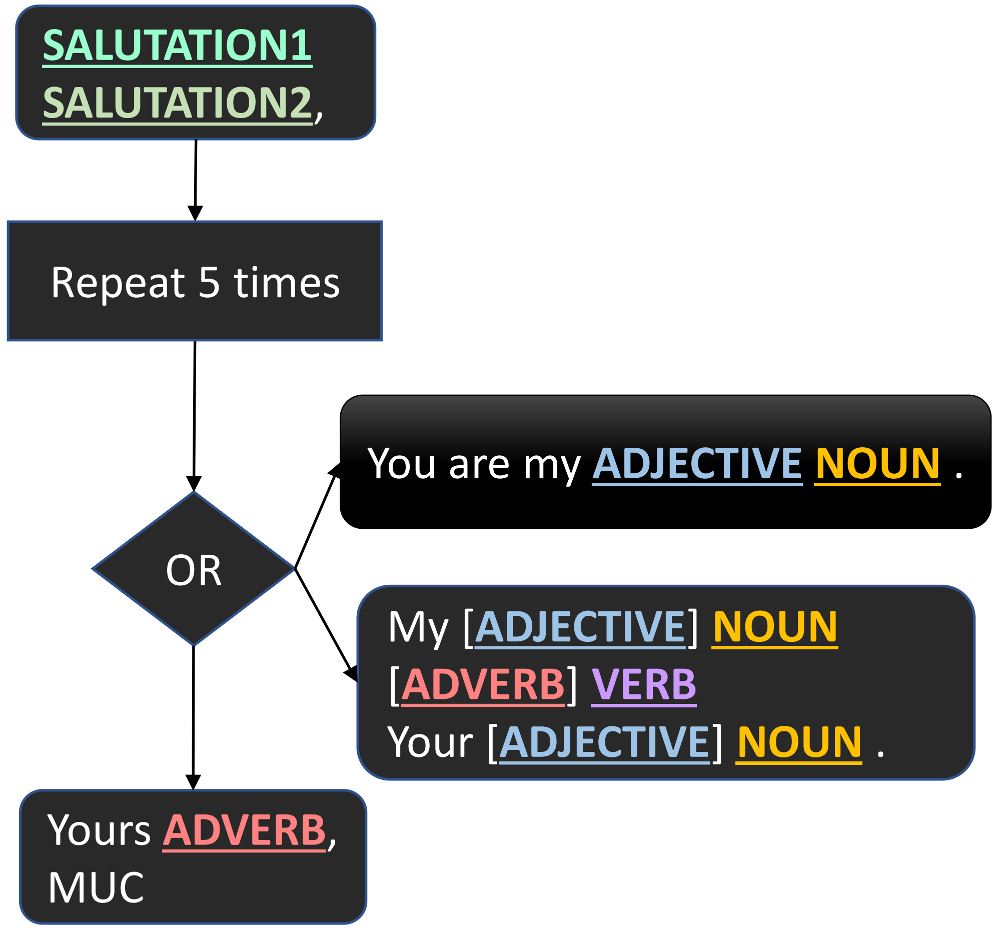
- D = get_discriminant(A,B,C)
- If D is None → return empty list []
- If D == 0 → return – [-B /(2*A)]
- Else → Compute x1, and x2, and return [x1, x2]

Homework 2

Functions and algorithms

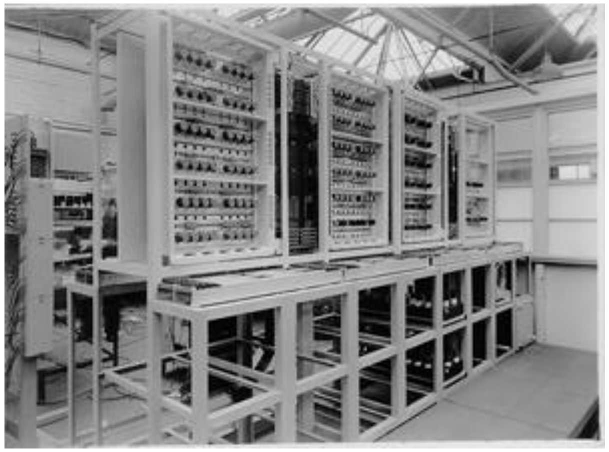
Homework 2: a little bit of history

- August 1953 – strange love letters appear on the notice board at the University of Manchester's computer lab
- They are all variations on a basic syntactic template:
- And the signatory is: "M.U.C." (Manchester University computer)



Ferranti Mark 1

- The computer



- The console

The world's first general-purpose and commercially available machine of its kind.

Others used it strictly for numerical calculations: analyzing weapons trajectories or seeking prime factors of huge numbers
Strachey was the first AI programmer, he taught the machine to play checkers ("draughts," in British). If M.U.C.'s opponent made too many mistakes, it would print: "I refuse to waste any more time. Go and play with a human being."

Combinatorial nature of love

- Strachey: *"There are many obvious imperfections in [my algorithm] (indeed very little thought went into its devising), and the fact that the vocabulary was largely based on Roget's Thesaurus lends a very peculiar flavor to the results."*
- The interesting thing was that a simple setup, using only **70** base words, could produce a combinatorial explosion of results—on the order of **three hundred billion** different love letters.

"Ultimately the software is based on a reductionist position vis-à-vis love and its expression. Love is regarded as a recombinatory procedure with recurring elements."

David Link "Archaeology of Algorithmic Artefacts"

String.format()

- Template = “Dear {}, would you come with me to the party today at {}. Please bring {} dollars for snacks”

PyCharm

- Time to use real IDE (Integrated Development Environment)
- Download and install *PyCharm* (Community Edition) for your operating system:

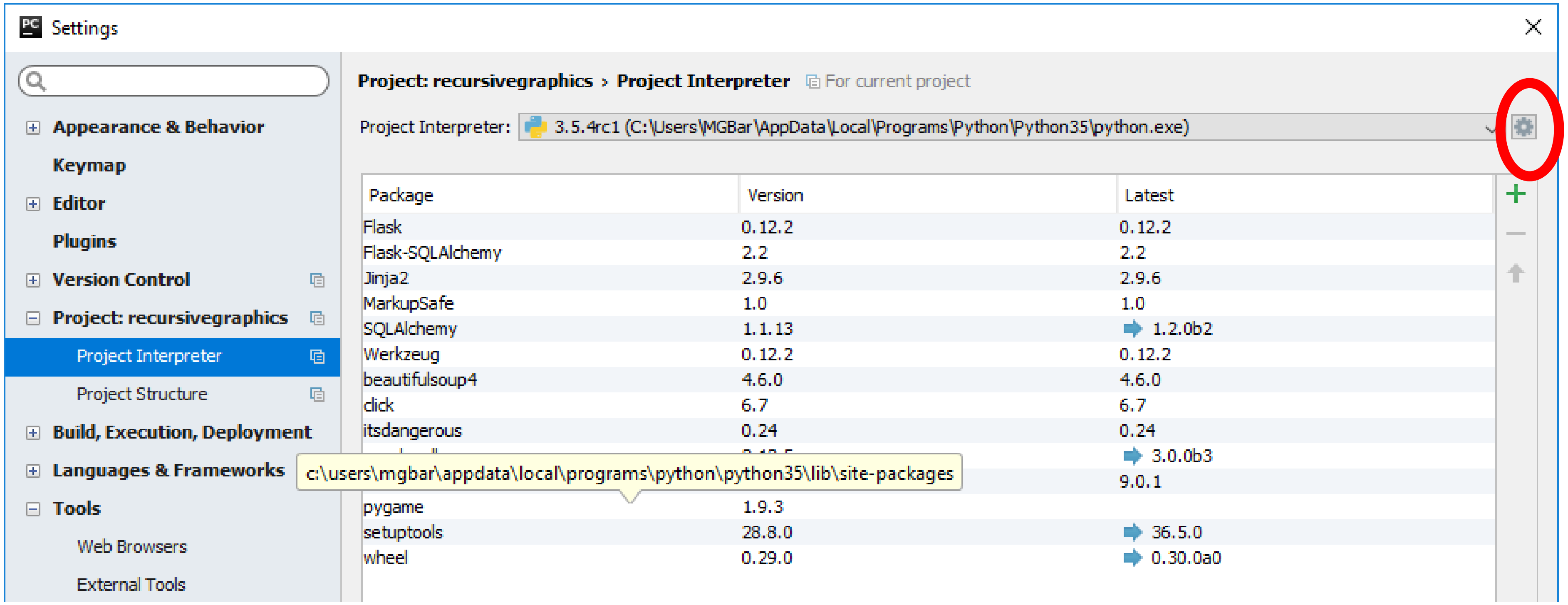
https://www.jetbrains.com/pycharm/download

Connect to 3.X version of Python
(previously installed)

File | Settings | Project Interpreter - for Windows and Linux

PyCharm | Preferences | Project Interpreter - for macOS

Ctrl + Alt + S



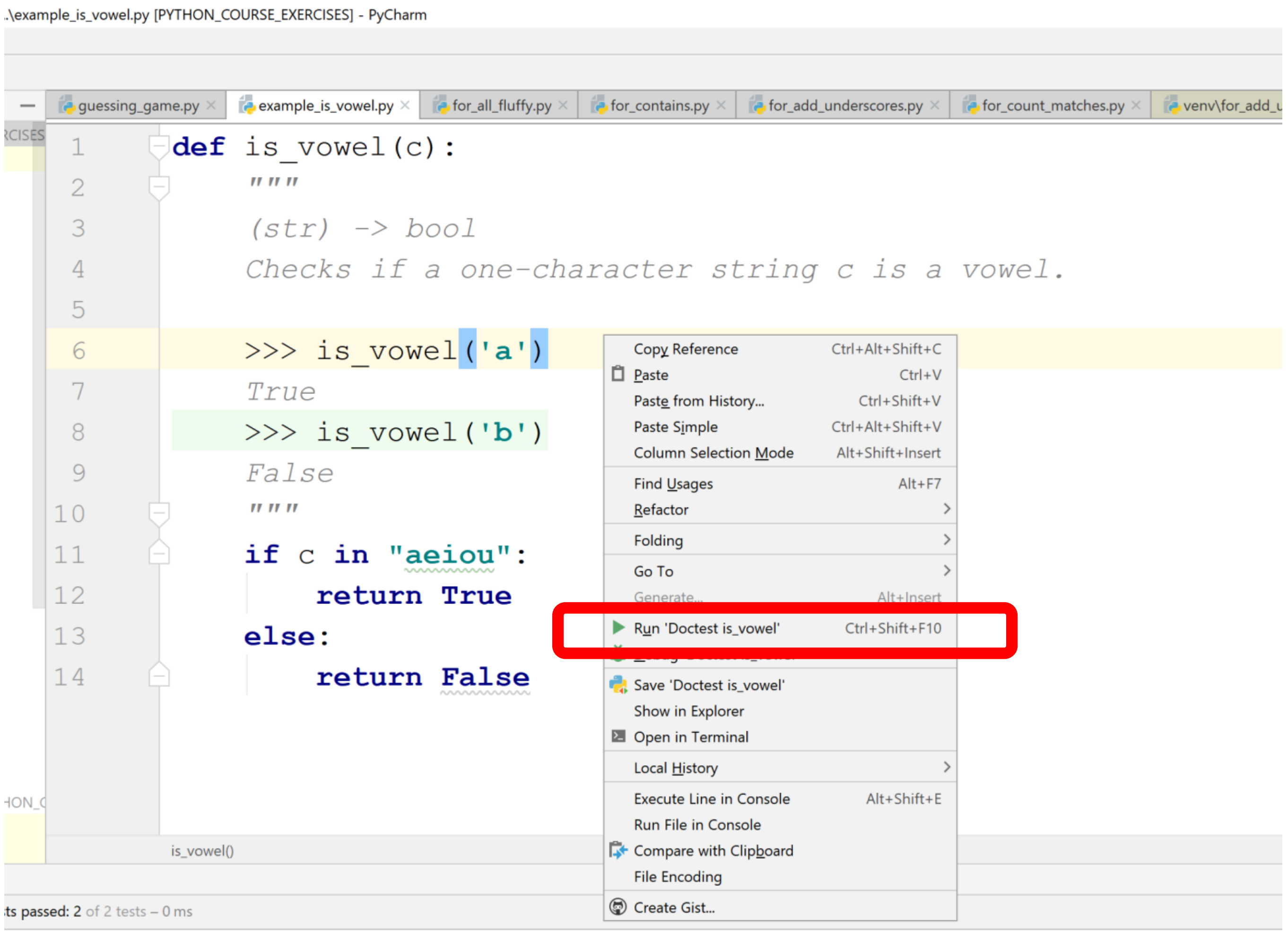
Project: recursivegraphics › Project Interpreter   For current project

Project Interpreter: 3.5.4rc1 (C:\Users\MGBar\AppData\Local\Programs\Python\Python35\python.exe)

Package	Version	Latest
Flask	0.12.2	0.12.2
Flask-SQLAlchemy	2.2	2.2
Jinja2	2.9.6	2.9.6
MarkupSafe	1.0	1.0
SQLAlchemy	1.1.13	➡ 1.2.0b2
Werkzeug	0.12.2	0.12.2
beautifulsoup4	4.6.0	4.6.0
click	6.7	6.7
itsdangerous	0.24	0.24
[illegible]	[illegible]	➡ 3.0.0b3
[illegible]	[illegible]	9.0.1
pygame	1.9.3	
setuptools	28.8.0	➡ 36.5.0
wheel	0.29.0	➡ 0.30.0a0

c:\users\mgbar\appdata\local\programs\python\python35\lib\site-packages

Using PyCharm for docstring tests



..\example_is_vowel.py [PYTHON_COURSE_EXERCISES] - PyCharm

guessing_game.py | example_is_vowel.py | for_all_fluffy.py | for_contains.py | for_add_underscores.py | for_count_matches.py | venv\for_add_u

```python
def is_vowel(c):
    """
    (str) -> bool
    Checks if a one-character string c is a vowel.

    >>> is_vowel('a')
    True
    >>> is_vowel('b')
    False
    """
    if c in "aeiou":
        return True
    else:
        return False
```

ts passed: 2 of 2 tests – 0 ms

Automatic testing

```
def is_vowel(c):
    """
    (str) -> bool
    Checks if a one-character string c is a vowel.

    >>> is_vowel('a')
    True
    >>> is_vowel('b')
    False
    """
    if c in "aeiou":
        return True
    else:
        return False
```

INDENT

SPACE after >>>