Learning Python

1 - Hello, World!

WELCOME - KNOWBIGDATA

- Expert Instructors
- CloudLabs
- Lifetime access to LMS
- Presentations
- Class Recording
- Assignments + Quizzes
- Project Work

- Real Life Project
- Course Completion Certificate
- 24x7 support
- KnowsBigData - Alumni
  - Jobs
  - Stay Abreast (Updated Content, Complimentary Sessions)
  - Stay Connected

ABOUT INSTRUCTOR - SANDEEP GIRI

2014	**KnowBigData**	Founded
2014	**Amazon**	Built High Throughput Systems for Amazon.com site using in-house NoSql.
2012	**InMobi**	Built Recommender that churns 200 TB
2011	**tBits Global**	Founded tBits Global Built an enterprise grade Document Management System
2006	**D.E.Shaw**	Built the big data systems before the term was coined
2002	**IIT Roorkee**	Finished B.Tech.

General Questions

Why Do People Use Python?

Why Do People Use Python?

1. Software quality

Readability => Reusable, Maintainable
Object-oriented (OO)
Functional

2. Developer productivity

Dynamic Types
Code Size: 1/3 to 1/5 of C++ or Java code.
Short Code => Less to type, debug, maintain

Why Do People Use Python?

3. Program portability

Same program runs on windows, linux and mac

4. Support libraries

Standard library

text pattern matching to network scripting

Third-party

+ Website construction
+ Numeric programming
+ Serial port access
+ Game development
+ (e.g.) NumPy is better than Matlab

Why Do People Use Python?

Component Integration

- Can invoke C and C++ libraries
- Can be called from C and C++
- Can integrate with Java and .NET, COM and Silverlight,
- Can interface with devices over serial ports
- Interact over networks with interfaces like SOAP, XML-RPC, and CORBA.

Enjoyment

- Act of programming more pleasure than chore

Is it scripting Language?

Is it scripting Language?

Yes, general-purpose programming language that blends procedural, functional, and object-oriented paradigms

What is downside?

- **Execution speed - lower than C/C++**
  - Source Code => byte code => execution
  - You can use PyPy to compile & speed up by 10x-100x
  - You can also link the compiled extension for Numeric

Who is using Python?

Google
Tube
NETFLIX
intel inside
Dropbox
INDUSTRIAL
LIGHT & MAGIC
A LUCASFILM COMPANY
CISCO

Who is using Python?

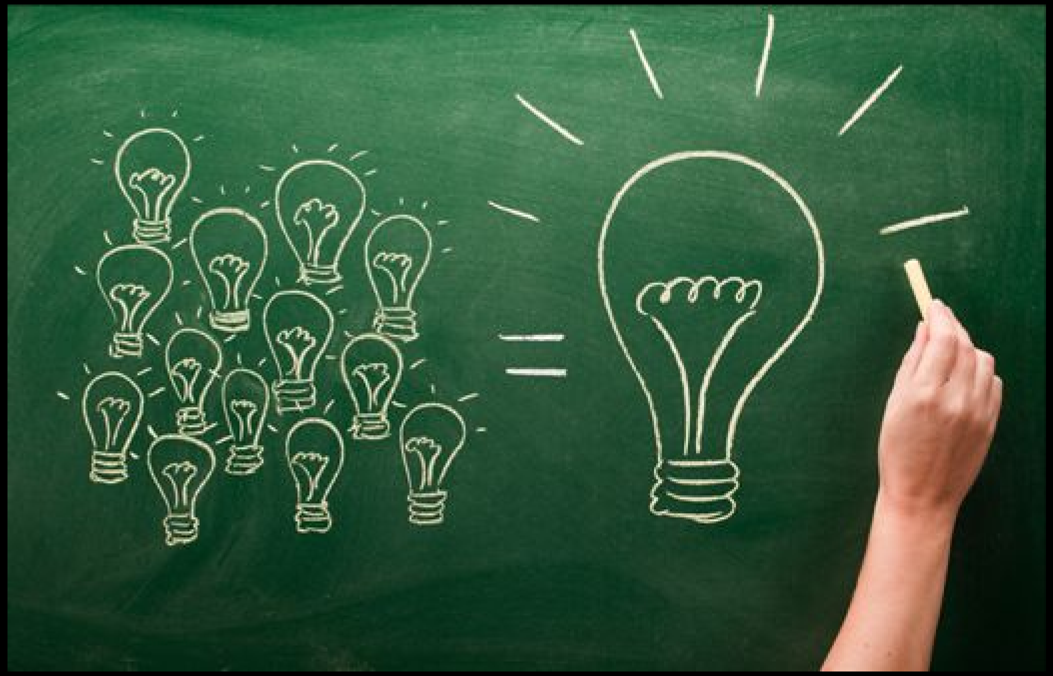
- **Success stories:** http://www.python.org/about/success
- **Application domains:** http://www.python.org/about/apps
- **User quotes:** http://www.python.org/about/quotes
- **Wikipedia page:** http://en.wikipedia.org/wiki/List_of_Python_software

What Can I Do with Python?

What Can I Do with Python?

- Systems Programming
- GUIs
- Internet Scripting
- Component Integration
- Database Programming
- Rapid Prototyping
- Numeric and Scientific Programming
- And More: Gaming, Images, Data Mining, Robots, Excel...

How Is IT Developed & Supported?

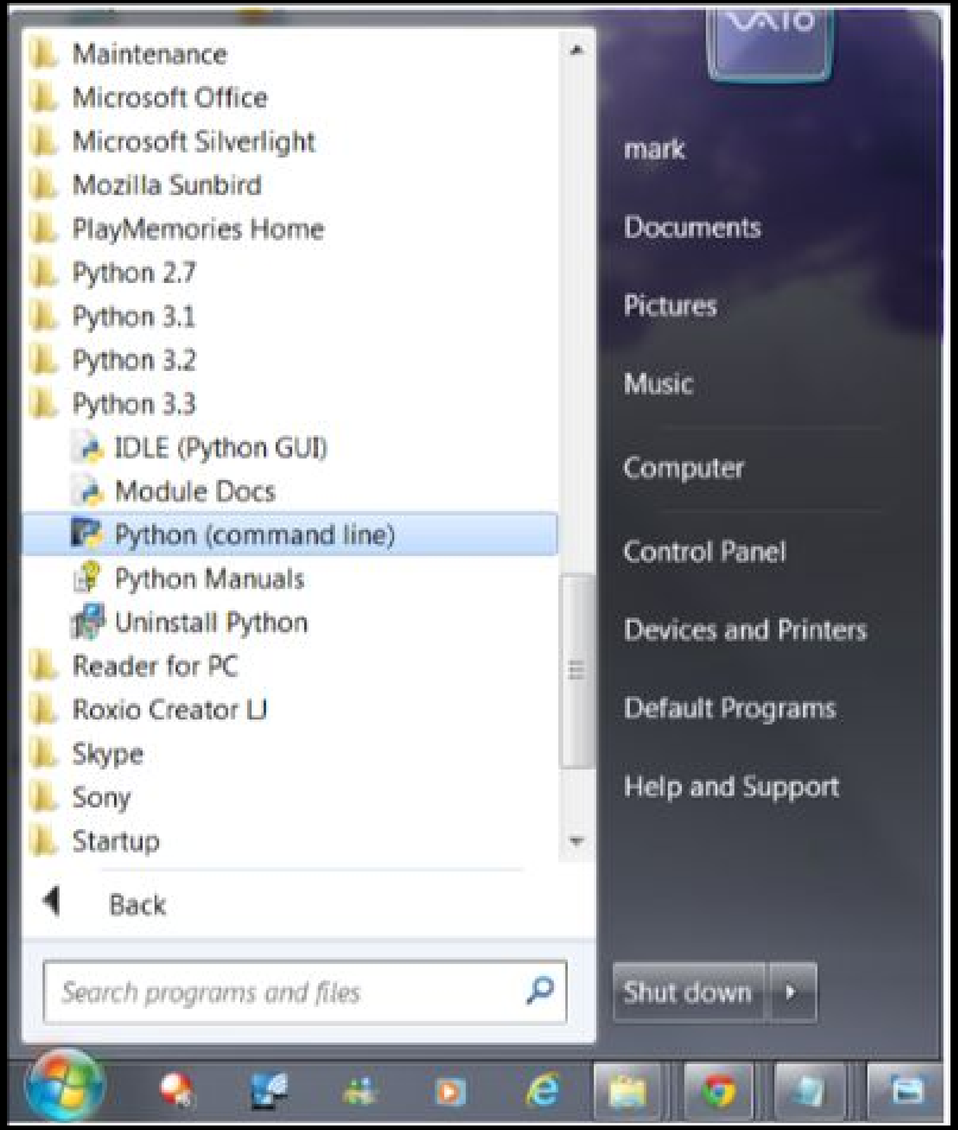


open source™

- Python Software Foundation
- PyCon
- Python Enhancement Proposal

Python Interpreter

1. Download from https://www.python.org/downloads/
2. Start the command line interpreter



```
Python 2.7.8 (v2.7.8:ee879c0ffa11, Jun 29 2014, 21:07:35)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello, World!"
Hello, World!
>>> 1+2**5
33
>> exit()
```

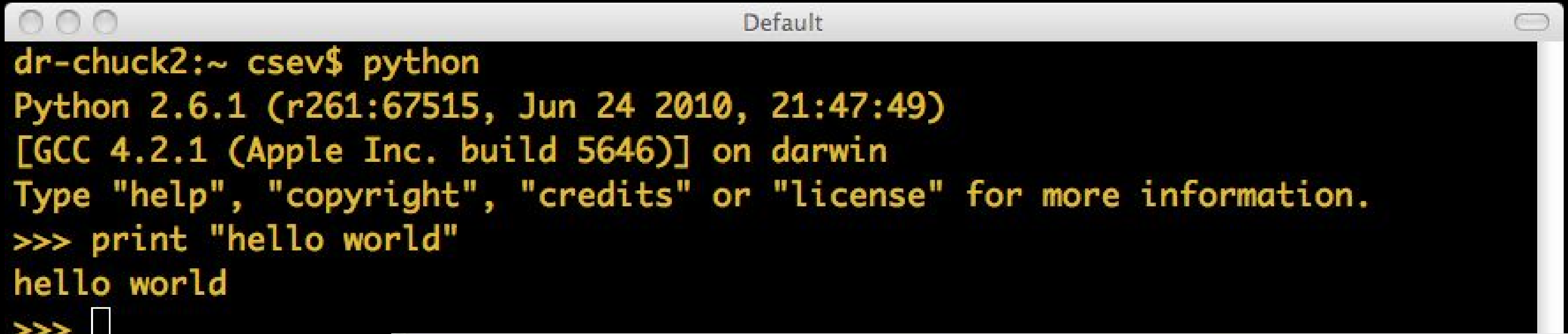
This is a good test to make sure that you have Python correctly installed.

Python Interpreter

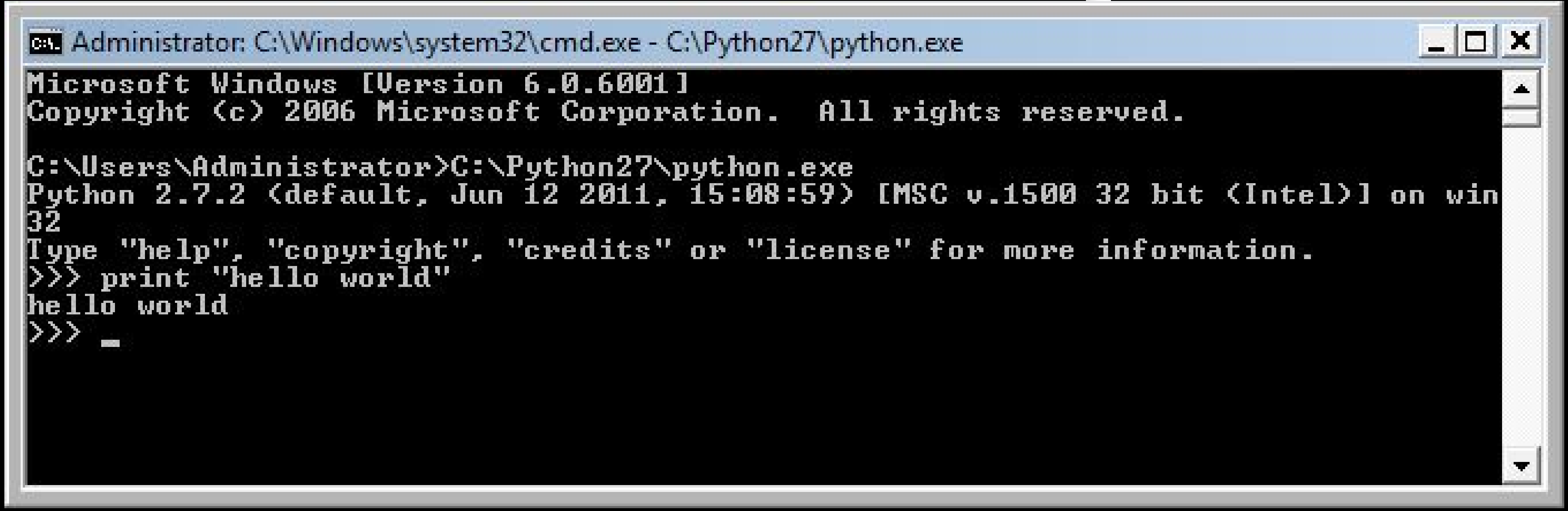
Why?

1. Very Quick to test
2. Good for trying things & Learning
3. Works like a shell or command prompt for scripting needs
4. Quick Automation

Let’s Talk to Python...



```
dr-chuck2:~ csev$ python
Python 2.6.1 (r261:67515, Jun 24 2010, 21:47:49)
[GCC 4.2.1 (Apple Inc. build 5646)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print "hello world"
hello world
>>> 
```



Administrator: C:\Windows\system32\cmd.exe - C:\Python27\python.exe

```
Microsoft Windows [Version 6.0.6001]
Copyright (c) 2006 Microsoft Corporation.  All rights reserved.

C:\Users\Administrator>C:\Python27\python.exe
Python 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> print "hello world"
hello world
>>> _
```

Elements of Python

- Vocabulary / Words - Variables and Reserved words (Chapter 2)
- Sentence structure - valid syntax patterns (Chapters 3-5)
- Story structure - constructing a program for a purpose

A short “story” about how to count words in a file in Python

```
name = raw_input('Enter file:')
handle = open(name, 'r')
text = handle.read()
words = text.split()

counts = dict()
for word in words:
   counts[word] = counts.get(word,0) + 1
bigcount = None
bigword = None

for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count
print bigword, bigcount
```

```
python words.py
Enter file: words.txt
to 16
```

Reserved Words

- You can not use reserved words as variable names / identifiers

and del for is raise assert elif from
lambda return break else global
not try class except if or while
continue exec import pass yield
def finally in print as with

Sentences or Lines

`x = 2`	← Assignment statement
`x = x + 2`	← Assignment with expression
`print x`	← Print statement

Variable Operator Constant Reserved Word

Writing a Simple Program

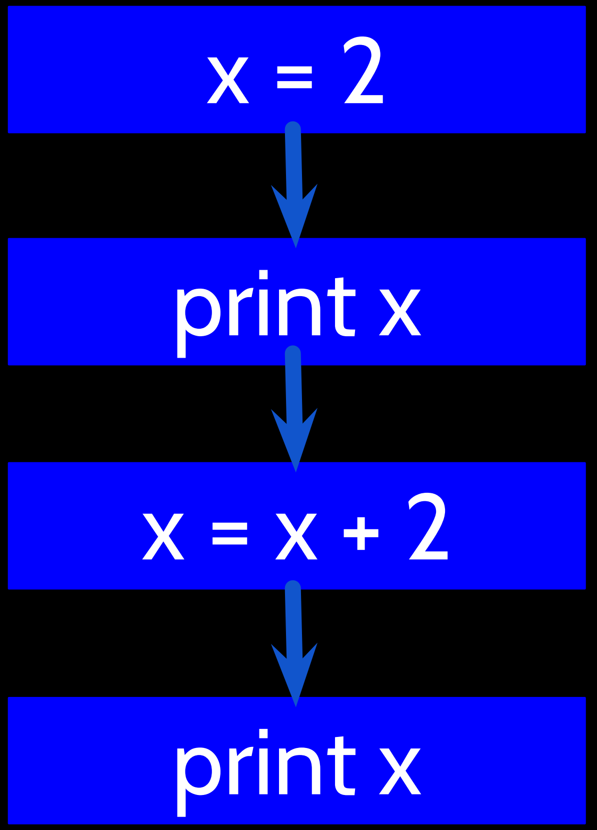
Interactive versus Script

- Interactive
  - You type directly to Python one line at a time and it responds
- Script
  - You enter a sequence of statements (lines) into a file using a text editor and tell Python to execute the statements in the file

Program Steps or Program Flow

- Like a recipe or installation instructions, a program is a sequence of steps to be done in order
- Some steps are conditional - they may be skipped
- Sometimes a step or group of steps are to be repeated
- Sometimes we store a set of steps to be used over and over as needed several places throughout the program (Chapter 4)

Sequential Steps



Program:

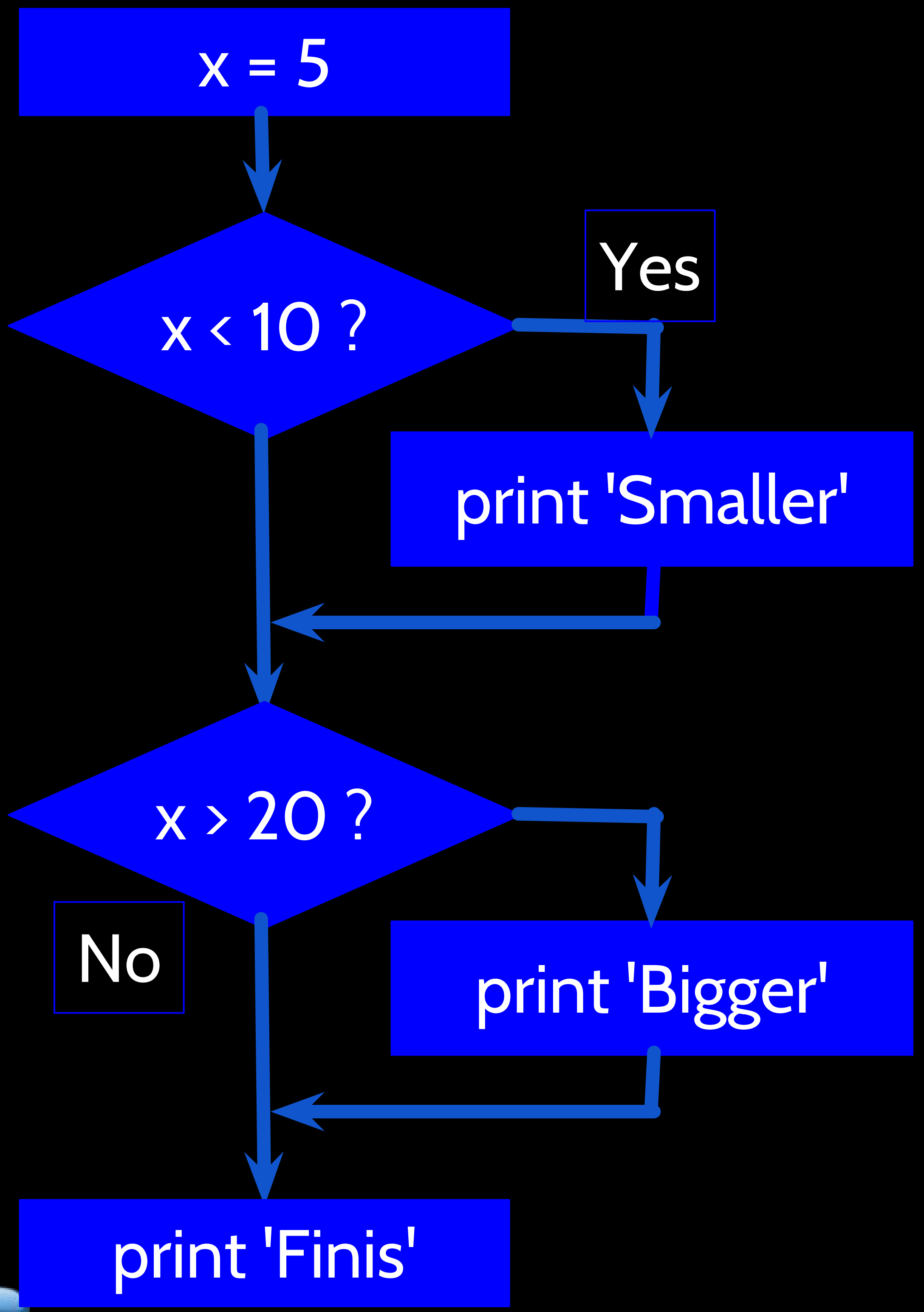
```
x = 2
print x
x = x + 2
print x
```

Output:

```
2
4
```

When a program is running, it flows from one step to the next. As programmers, we set up “paths” for the program to follow.

Conditional Steps



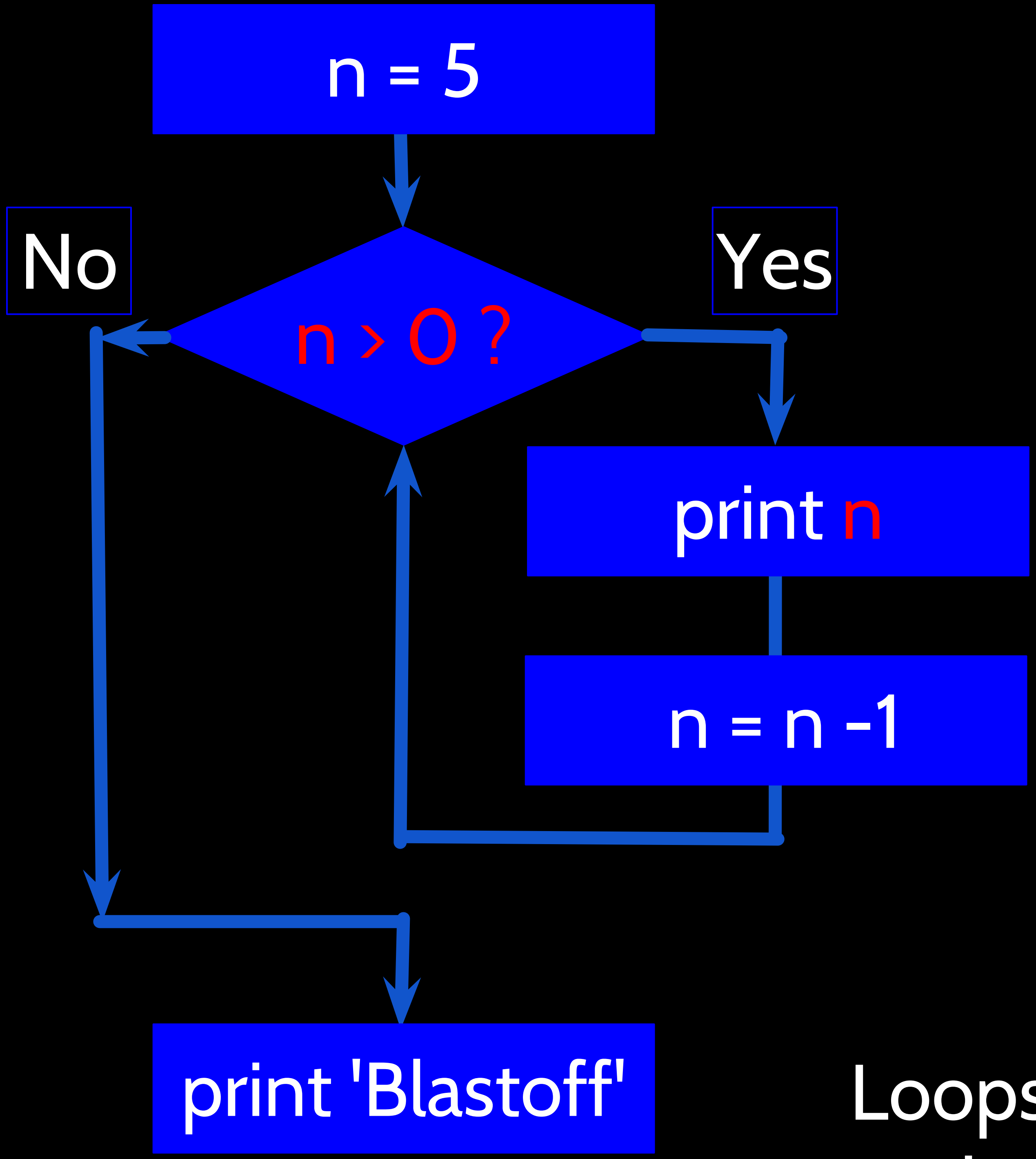
Program:

```
x = 5
if x < 10:
    print 'Smaller'
if x > 20:
    print 'Bigger'

print 'Finis'
```

Output:

```
Smaller
Finis
```



Repeated Steps

Program:

```
n = 5
while n > 0 :
    print n
    n = n - 1
print 'Blastoff!'
```

Output:

```
5
4
3
2
1
Blastoff!
```

Loops (repeated steps) have iteration variables that change each time through a loop. Often these iteration variables go through a sequence of numbers.

```
name = raw_input('Enter file:')
handle = open(name, 'r')
text = handle.read()
words = text.split()

counts = dict()
for word in words:
   counts[word] = counts.get(word,0) + 1
bigcount = None
bigword = None

for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print bigword, bigcount
```

Sequential

Repeated

Conditional

```
name = raw_input('Enter file:')
handle = open(name, 'r')
text = handle.read()
words = text.split()
counts = dict()
for word in words:
   counts[word] = counts.get(word,0) + 1

bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > 
bigcount:
        bigword = word
        bigcount = count

print bigword, bigcount
```

A short Python “Story” about how to count words in a file

A word used to read data from a user

A sentence about updating one of the many counts

A paragraph about how to find the largest item in a list

Python

www.KnowBigData.com

Know BIG DATA

Summary

- This is a quick overview
- We will revisit these concepts throughout the course
- Focus on the big picture

Questions?

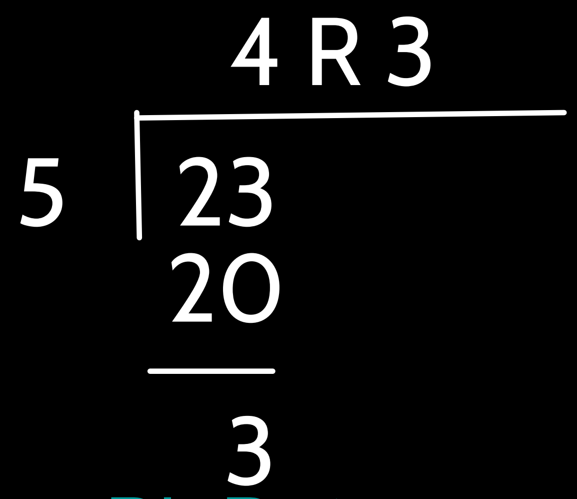
Learning Python

2. Variables, Expressions, & Statements

Numeric Expressions

```
>>> xx = 2
>>> xx = xx + 2
>>> print xx
4
>>> yy = 440 * 12
>>> print yy
5280
>>> zz = yy / 1000
>>> print zz
5
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print kk
3
>>> print 4 ** 3
64
```



Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Type Matters

- Python knows what "type" everything is
- Some operations are prohibited
- You cannot "add 1" to a string
- We can ask Python what type something is by using the type() function

```
>>> eee = 'hello ' + 'there'
>>> eee = eee + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
TypeError: cannot concatenate
'str' and 'int' objects
>>> type(eee)
<type 'str'>
>>> type('hello')
<type 'str'>
>>> type(1)
<type 'int'>
>>>
```

Several Types of Numbers

- Numbers have two main types
  - Integers are whole numbers: -14, -2, 0, 1, 100, 401233
  - Floating Point Numbers have decimal parts: -2.5 , 0.0, 98.6, 14.0
- There are other number types - they are variations on float and integer

```
>>> xx = 1
>>> type (xx)
<type 'int'>
>>> temp = 98.6
>>> type(temp)
<type 'float'>
>>> type(1)
<type 'int'>
>>> type(1.0)
<type 'float'>
>>>
```

Type Conversions

- When you put an integer and floating point in an expression, the integer is implicitly converted to a float

- You can control this with the built-in functions int() and float()

```
>>> print float(99) / 100
0.99
>>> i = 42
>>> type(i)
<type 'int'>
>>> f = float(i)
>>> print f
42.0
>>> type(f)
<type 'float'>
>>> print 1 + 2 * float(3) / 4 - 5
-2.5
>>>
```

User Input

- We can instruct Python to pause and read data from the user using the raw_input() function
- The raw_input() function returns a string

```
nam = raw_input('Who are you?')
print 'Welcome', nam
```

```
Who are you? Chuck
Welcome Chuck
```

Comments in Python

- Anything after a # is ignored by Python
- ''' for block comments

- Why comment?
  - Describe what is going to happen in a sequence of code
  - Document who wrote the code or other ancillary information
  - Turn off a line of code - perhaps temporarily

Indentation

- Increase indent indent after an if statement or for statement (after :)
- Maintain indent to indicate the scope of the block (which lines are affected by the if/for)
- Reduce indent *back to* the level of the if statement or for statement to indicate the end of the block
- Blank lines are ignored - they do not affect indentation
- Comments on a line by themselves are ignored with regard to indentation

Exercise

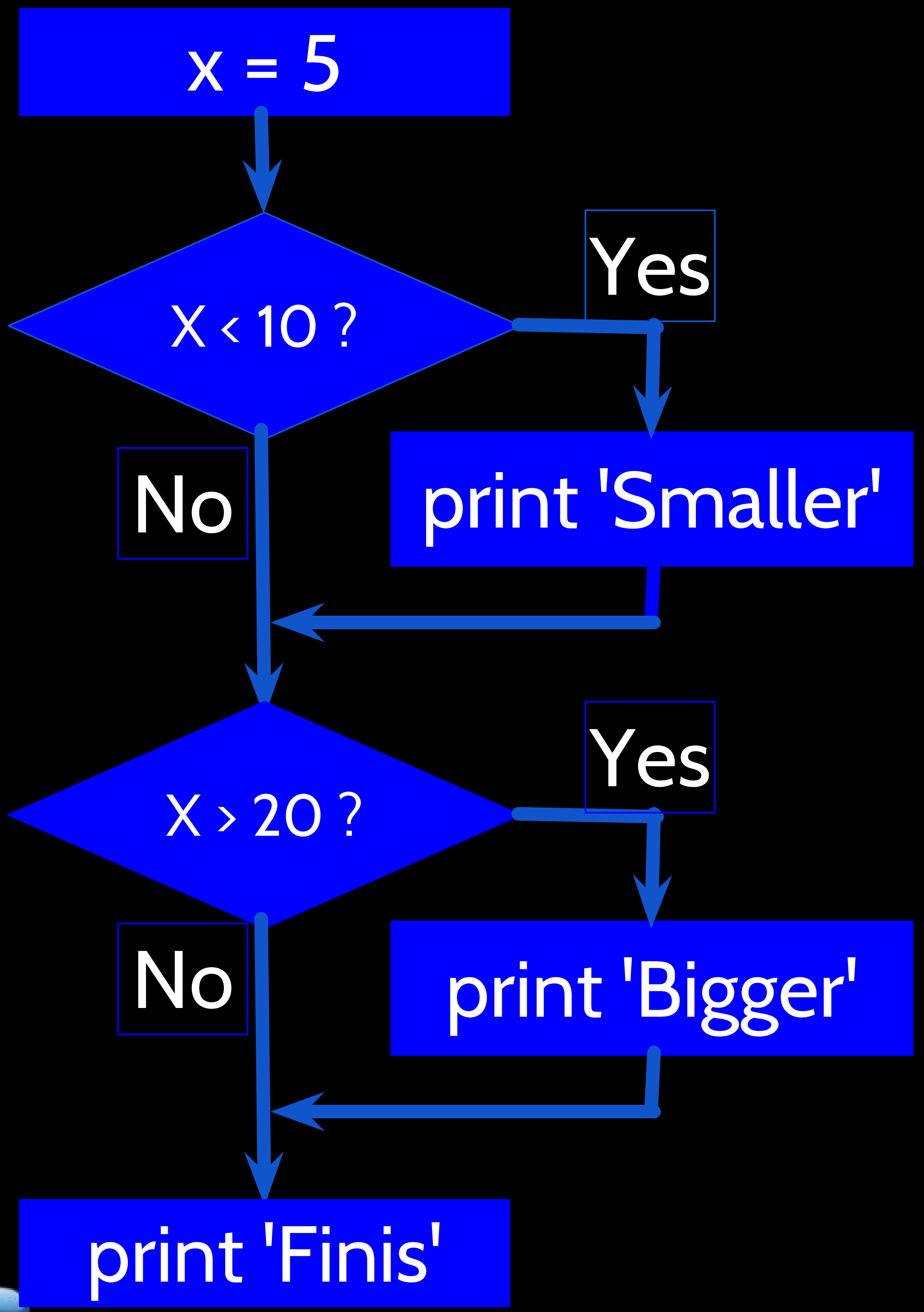
Write a program to prompt the user for hours and rate per hour to compute gross pay.

```
Enter Hours: 35
Enter Rate: 2.75
Pay: 96.25
```

Learning Python

Session 3 - Conditional Execution

Conditional Steps



x = 5

X < 10 ?

Yes

print 'Smaller'

No

X > 20 ?

Yes

print 'Bigger'

No

print 'Finis'

Program:

```
x = 5
if x < 10:
   print 'Smaller’

if x > 20:
     print 'Bigger'

print 'Finis'
```

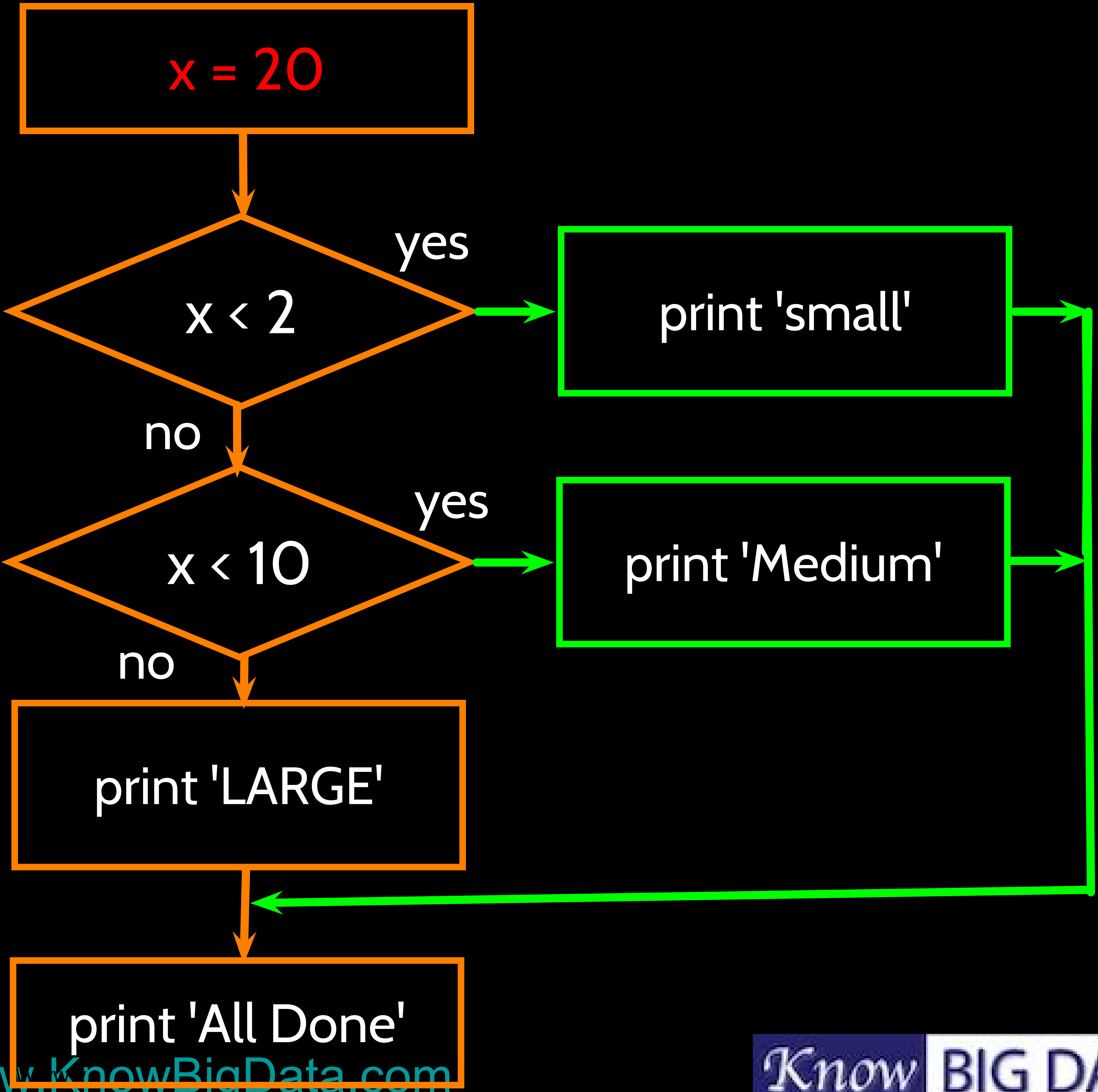
Output:

```
Smaller
Finis
```

Python

Multi-way

```
x = 20
if x < 2 :
    print 'small'
elif x < 10 :
    print 'Medium'
else :
    print 'LARGE'
print 'All done'
```



Python

www.KnowBigData.com

Know BIG DATA

The try / except Structure

- You surround a dangerous section of code with try and except
- If the code in the try works - the except is skipped
- If the code in the try fails - it jumps to the except section

```
$ cat tryexcept.py
astr = 'Hello Bob'
try:
    istr = int(astr)
except:
    istr = -1

print 'First', istr

astr = '123'
try:
    istr = int(astr)
except:
    istr = -1

print 'Second', istr
```

When the first conversion fails - it just drops into the except: clause and the program continues.

```
$ python tryexcept.py
First -1
Second 123
```

When the second conversion succeeds - it just skips the except: clause and the program continues.

Sample try / except

```
rawstr = raw_input('Enter a number:')
try:
    ival = int(rawstr)
except:
    ival = -1

if ival > 0 :
    print 'Nice work'
else:
    print 'Not a number'
```

```
$ python trynum.py
Enter a number:42
Nice work
$ python trynum.py
Enter a number:forty-two
Not a number
$
```

Exercise

Rewrite your pay computation to give the employee 1.5 times the hourly rate for hours worked above 40 hours.

```
Enter Hours: 45
Enter Rate: 10
Pay: 475.0
```

$475 = 40 * 10 + 5 * 15$

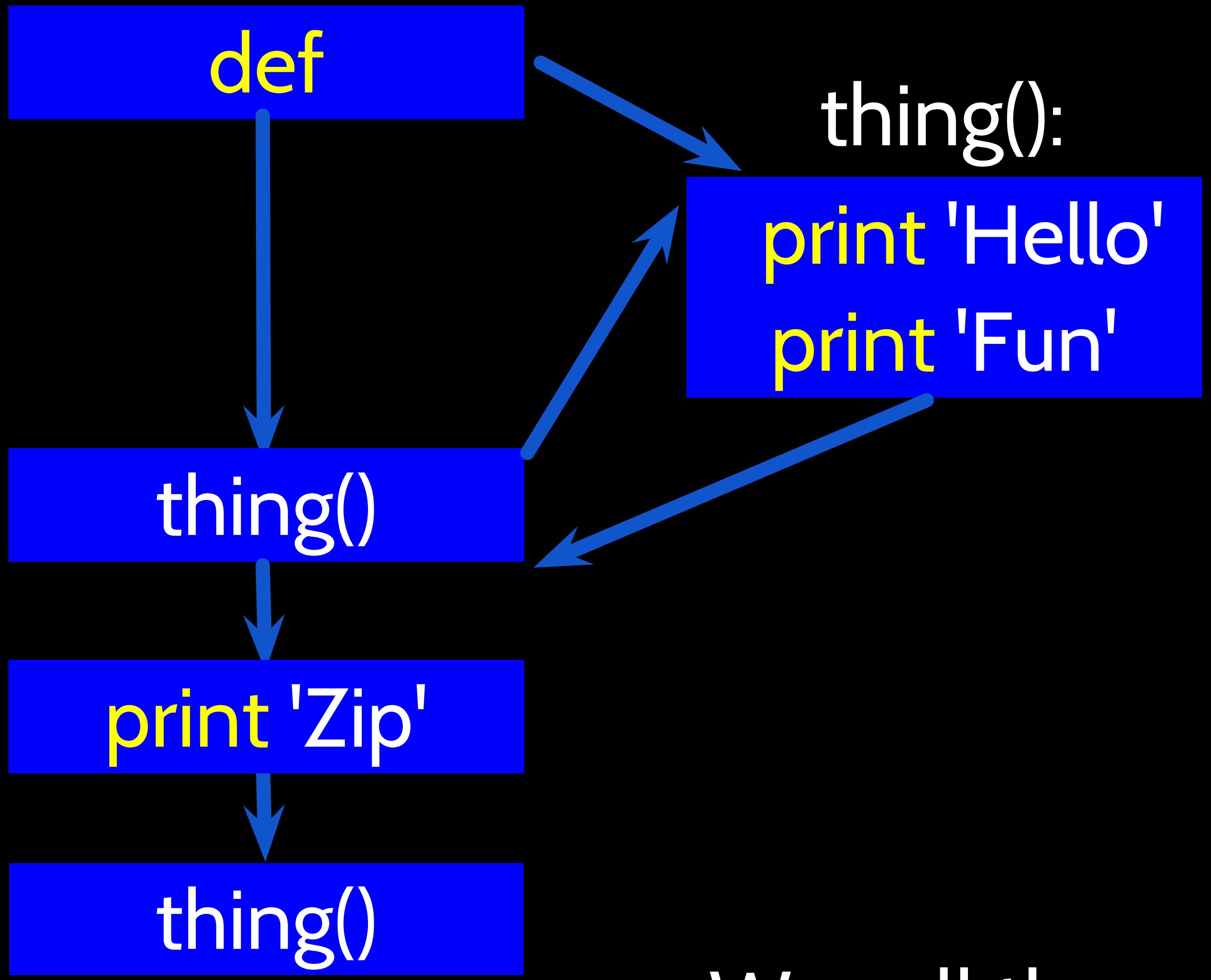
Summary

- Comparison operators
  == <= >= > < ! =
- Logical operators: and or not
- Indentation
- One-way Decisions
- Two-way decisions:
  if: and else:
- Nested Decisions
- Multi-way decisions using elif
- Try / Except to compensate for errors
- Short circuit evaluations

Learning Python

Session 4 - Functions

Stored (and reused) Steps



Program:

```
def thing():
    print 'Hello'
    print 'Fun'

thing()
print 'Zip'
thing()
```

Output:

```
Hello
Fun
Zip
Hello
Fun
```

We call these reusable pieces of code “functions”

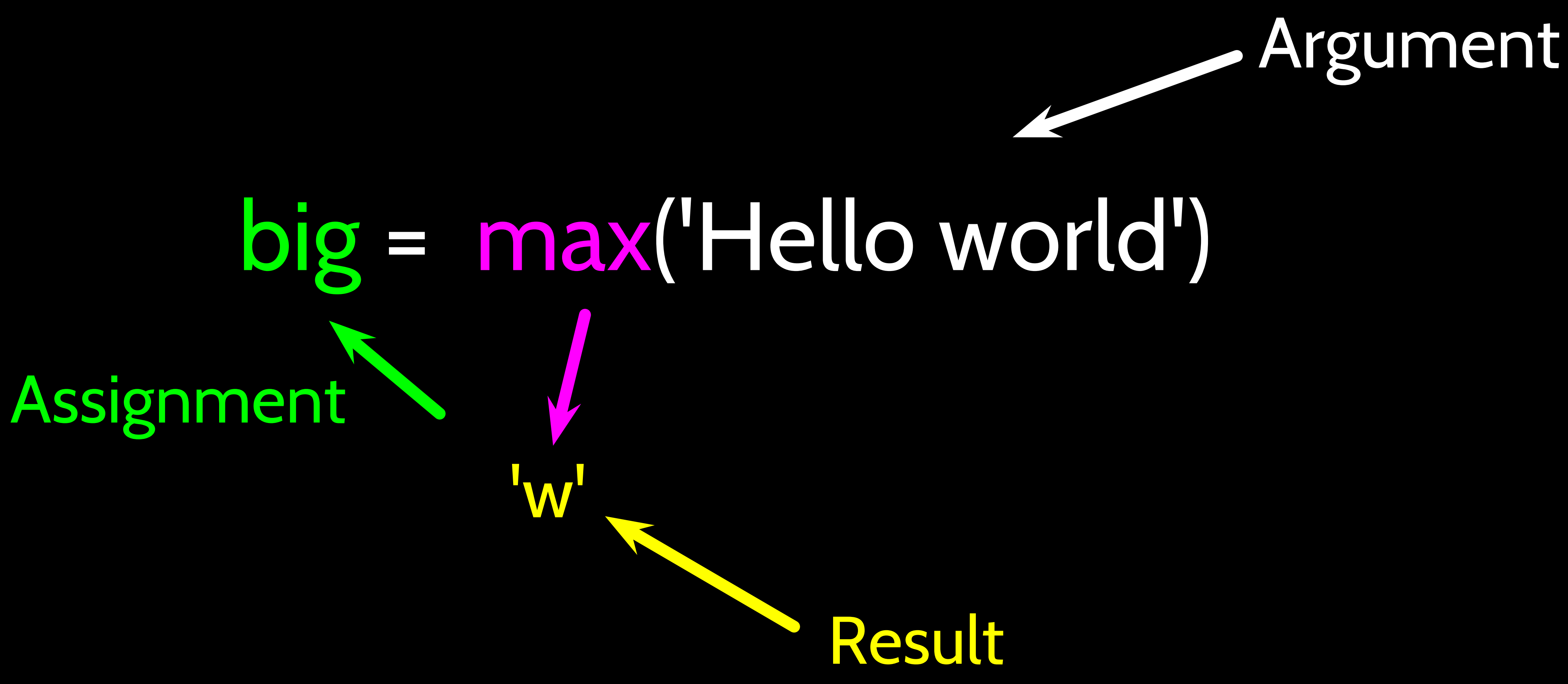
Python

www.KnowBigData.com

Know BIG DATA

Function Definition

- In Python a function is some reusable code that takes arguments(s) as input, does some computation, and then returns a result or results
- We define a function using the def reserved word
- We call/invoke the function by using the function name, parentheses, and arguments in an expression



Argument

big = max('Hello world')

Assignment

'w'

Result

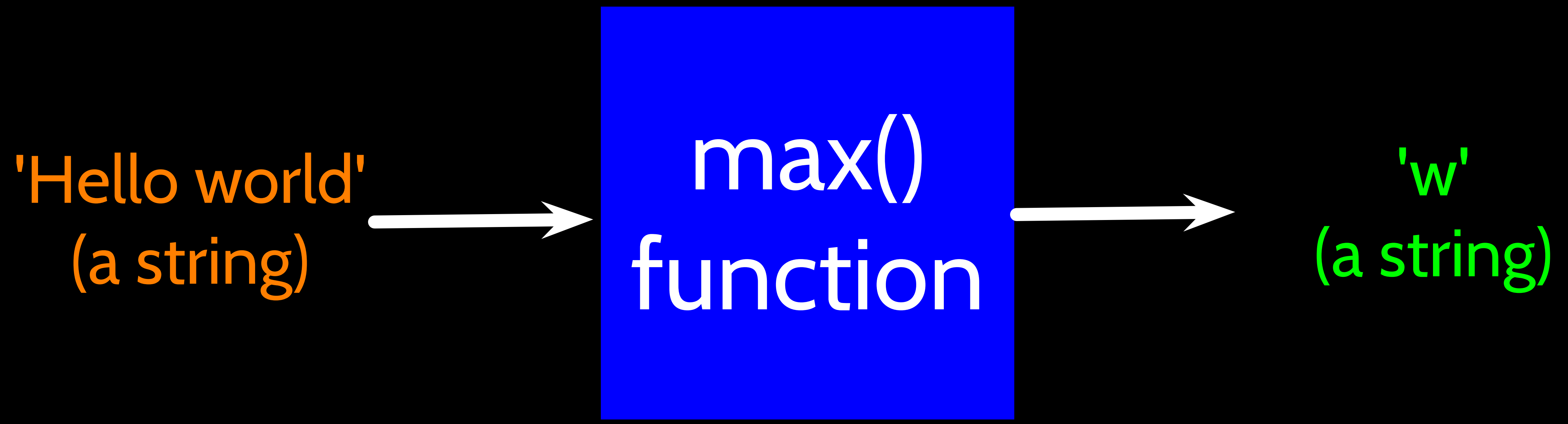
```
>>> big = max('Hello world')
>>> print big
w
>>> tiny = min('Hello world')
>>> print tiny

>>>
```

Max Function

```
>>> big = max('Hello world')
>>> print big
w
```

A function is some stored code that we use. A function takes some input and produces an output.



'Hello world'
(a string)

max()
function

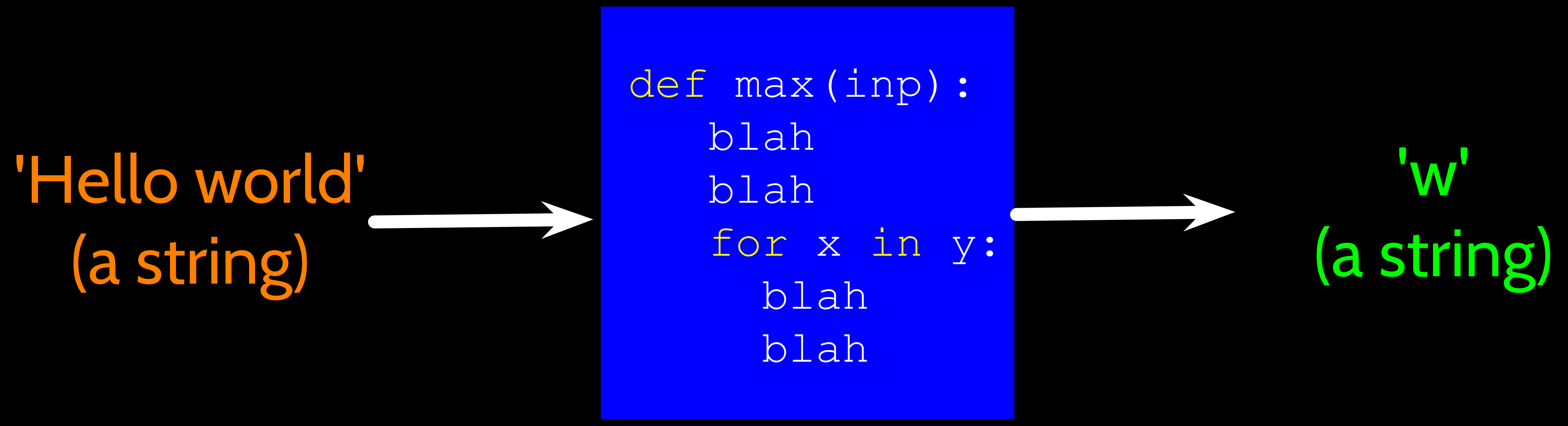
'w'
(a string)

Guido wrote this code

Max Function

```
>>> big = max('Hello world')
>>> print big
w
```

A function is some stored code that we use. A function takes some input and produces an output.



'Hello world'
(a string)

```
def max(inp):
   blah
   blah
   for x in y:
     blah
     blah
```

'w'
(a string)

Guido wrote this code

Variable Arguments

```
def manyArgs(*arg):
   print "I was called with", len(arg), "arguments:", arg

>>> manyArgs(1)
I was called with 1 arguments: (1,)
>>> manyArgs(1, 2,3)
I was called with 3 arguments: (1, 2, 3)
```

Variable Keyworded Arguments

```
def greet_me(**kwargs):
    if kwargs is not None:
        for key, value in kwargs.iteritems():
            print "%s == %s" %(key,value)

>>> greet_me(name="yasoob", age="10")
name == yasoob

age == 10
```

Passing Functions: Filter

- Executes a function on each element of array
- If the function returns True
- Puts it in output
- Distributable paradigm

```
def isEven(x):
   return x % 2 == 0;


>>> filter(isEven, [1,2,3,4]);
[2, 4]
```

Passing Functions: Map

- Executes a function on each element of array
- Returns the array containing output
- Distributable paradigm

```
def my_map(x):
   return x * 2;
>>>arr = [1,2,3,4]
>>>map(my_map,arr );
[2, 4, 6, 8]
```

Passing Functions: Reduce

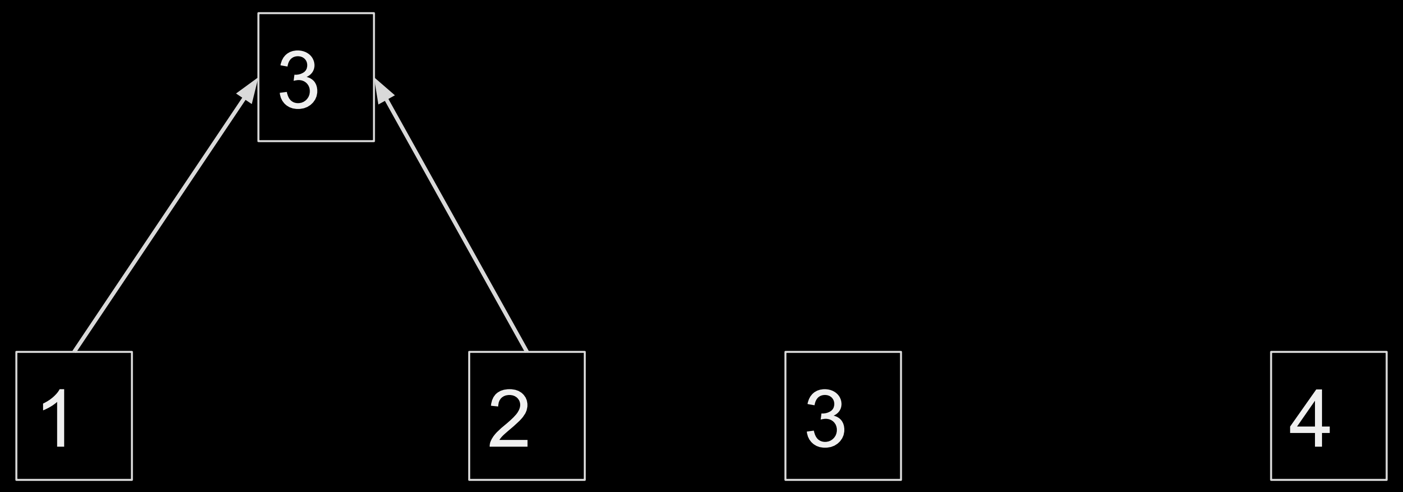
- Executes a function on two element of array
- Keeps executing recursively
- Distributable paradigm
- associative & cumulative

```
def my_sum(x, y):
   return x +y;
>>>arr = [1,2,3,4]
>>>reduce(my_sum,arr );
10
```

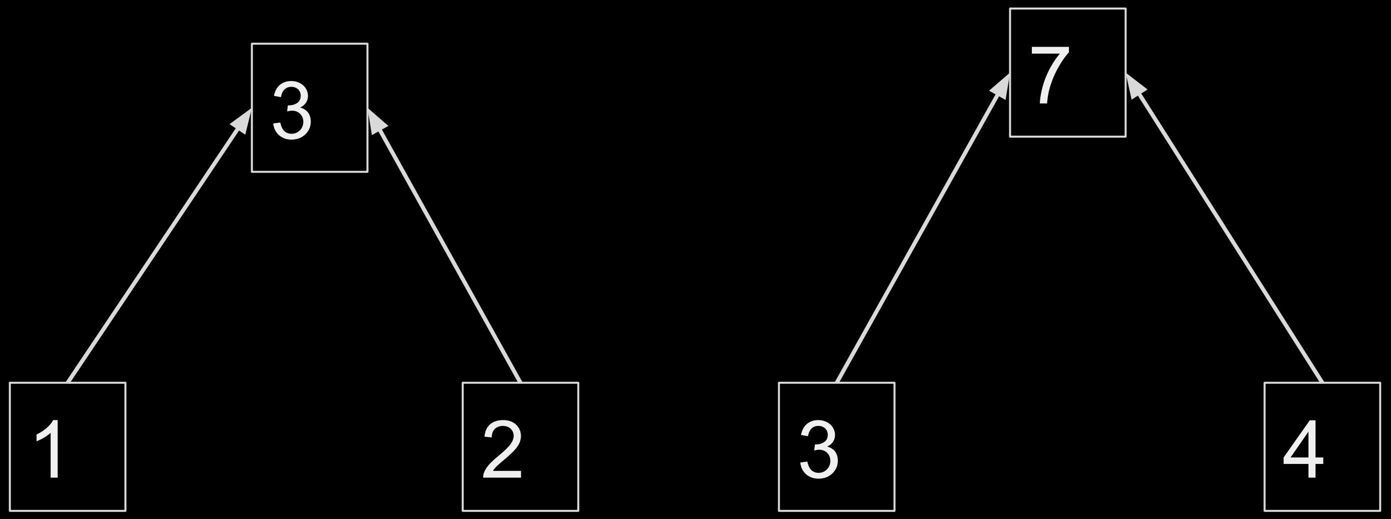
Cummulative & Associative

1. f(f(a,b) , c) == f(f(a,c), b) == f(f(b, c), a)
2. Example Addition function:
   a. (1 + 2) == (2 + 1)
   b. (1 + 2) + 3 == 1 + (2 + 3) == (3 + 1) + 2

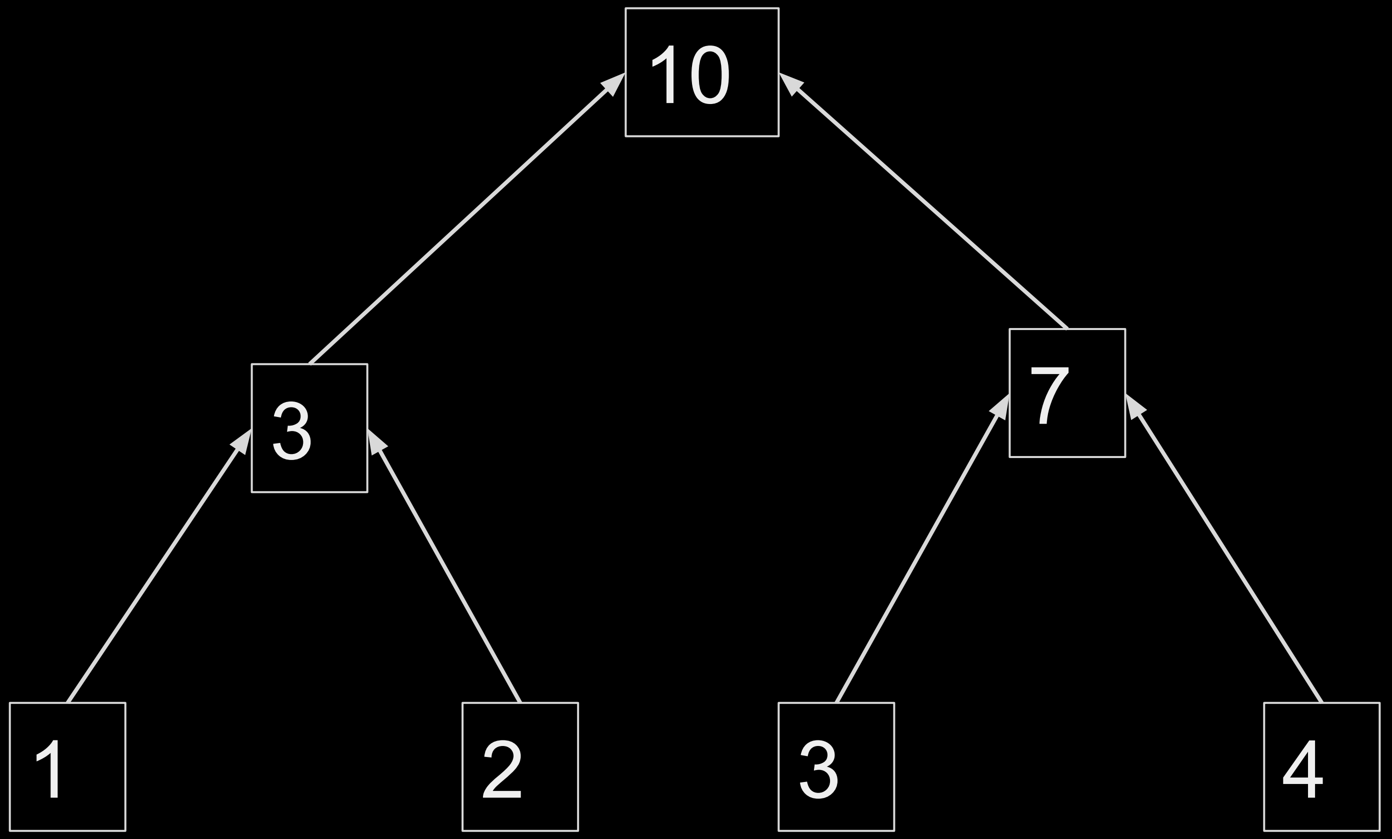
Passing Functions: Reduce



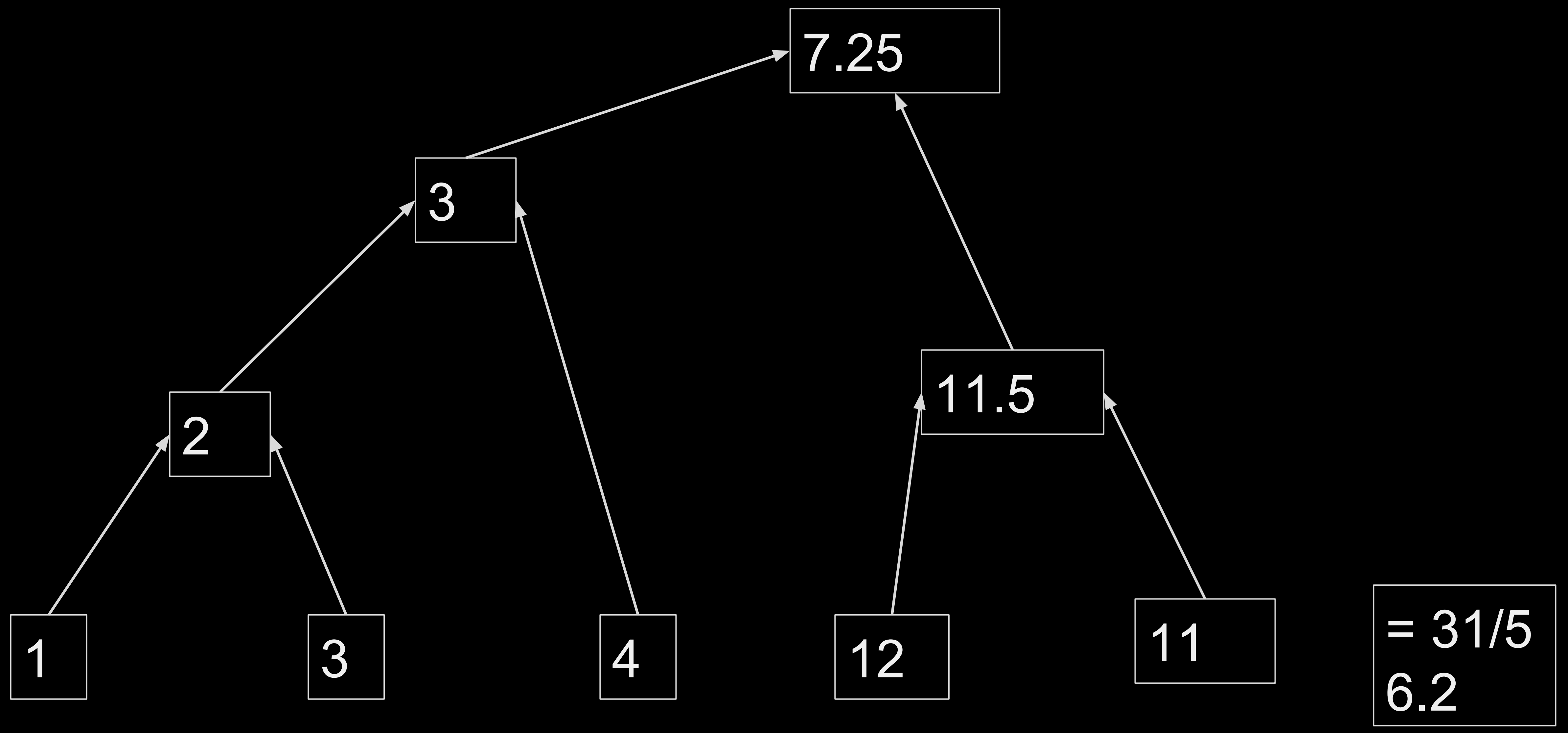
Passing Functions: Reduce



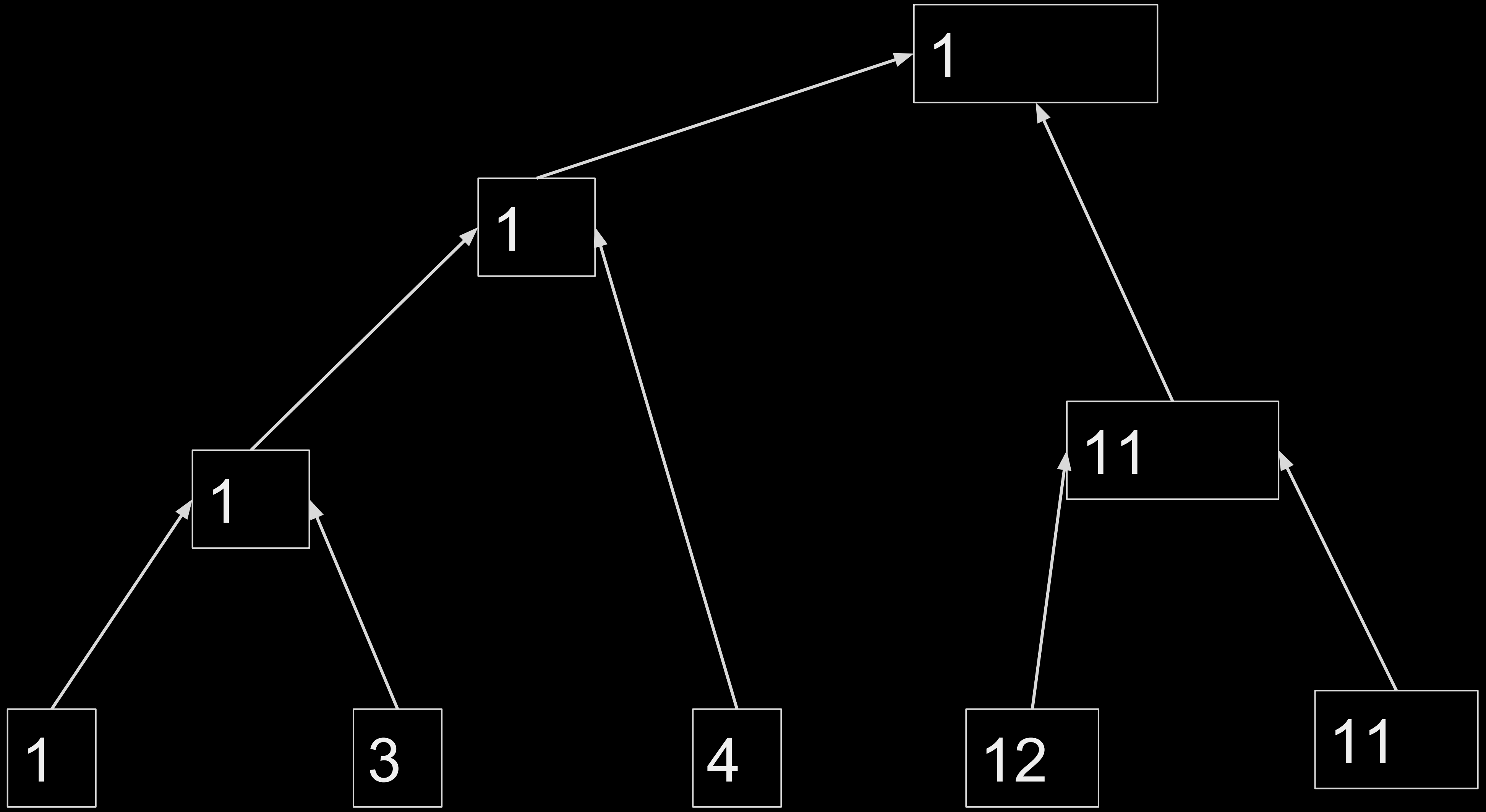
Passing Functions: Reduce



Passing Functions: Reduce - AVG



Passing Functions: Reduce - MIN



Q: Which of the following functions are okay for reduce?

1. Average
   ```
   def myfunc(x, y):
       return (x+y)/2.0;
   ```

2. Min(x,y)

3. Max(x,y)

4. def myfunct(x,y): return sqrt(x*x + y*y);

Q: Which of the following functions are okay for reduce?

✗ 1. Average

```
def myfunc(x, y):
        return (x+y)/2.0;
```

✓ 2. Min(x,y)

✓ 3. Max(x,y)

✓ 4. def myfunct(x,y): return sqrt(x*x + y*y);

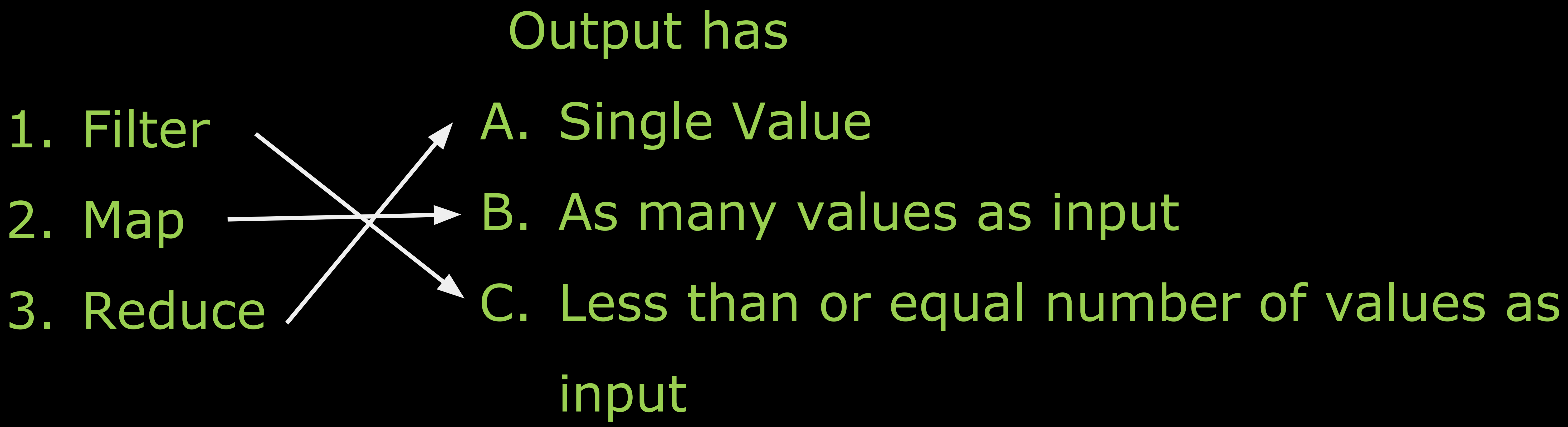
Match The results

1. Filter
2. Map
3. Reduce

Output has

A. Single Value
B. As many values as input
C. Less than or equal number of values as input

Match The results



Output has

1. Filter
2. Map
3. Reduce

A. Single Value

B. As many values as input

C. Less than or equal number of values as input

How are map(), reduce() and filter() distributable paradigms?

How are map(), reduce() and filter() distributable paradigms?

A. All can work on the range of the data
B. Divide & Conquer
C. The logic can travel

Lambda Function

1. Anonymous Function
2. Can be used quickly
3. Comes from functional programming

```
>>> def f (x): return x**2
>>> print f(8)
64
>>> g = lambda x: x**2
>>> print g(8)
64
```

Lambda Function: Map, Filter & Reduce

```
>>> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
>>> print filter(lambda x: x % 3 == 0, foo)
[18, 9, 24, 12, 27]
>>> print map(lambda x: x * 2 + 10, foo)
[14, 46, 28, 54, 44, 58, 26, 34, 64]
>>> print reduce(lambda x, y: x + y, foo)
```

Exercise

Rewrite your pay computation with time-and-a-half for overtime and create a function called computepay which takes two parameters (hours and rate).

```
Enter Hours: 45
Enter Rate: 10
Pay: 475.0
```

475 = 40 * 10 + 5 * 15

Summary

- Functions
- Built-In Functions
  - Type conversion (int, float)
  - String conversions
- Parameters
- Arguments
- Results (fruitful functions)
- Void (non-fruitful) functions
- Why use functions?