# Programming in Python

IAP 2013

Student Information Processing Board
Nathan Arce ‘13 – Luke O’Malley ‘14

# Day 1

1. Background
2. Install Python
3. Using Python as a Calculator
4. “Hello World!”
5. Functions and Arguments
6. Control Flow Tools
7. Coding Challenge

# Who Uses Python?


Google Search I'm Feeling Lucky

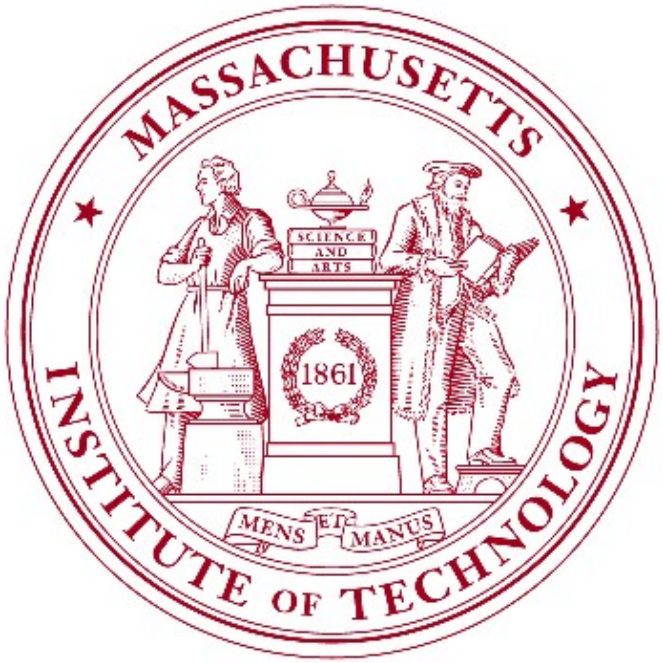
Dropbox

Watch a Video

Download Dropbox

Free for Windows, Mac, Linux, and Mobile


django


MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
SCIENCE
AND
ARTS
1861
MENS
ET
MANUS

# Why Use Python?

# Windows

1. Visit http://www.python.org/getit/
2. Install Python 2.7.3

# Mac/Linux

1. Check if Python is already installed by opening a Terminal
2. In the Terminal, type:

```
your-user-name$ python --version
Python 2.7.3
```

3. If it errors and you don't have Python call us over

# Use Python as a Calculator

“Hello World?”

# Functions and Arguments

# Conditional Flow Tools

# Coding Challenge!

# SIPB: Programming in Python, IAP 2013

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. Find the sum of all the multiples of 3 or 5 below 1000.

# End Day 1!

# SIPB: Programming in Python, IAP 2013

http://bit.ly/11a8ITS

# Programming in Python

IAP 2013

Student Information Processing Board
Nathan Arce ‘13 – Luke O’Malley ‘14

# Day 2

1. Coding Warm-up
2. Data Stores
    - Lists
    - Dictionaries
3. Objects/Classes
4. Recursive Functions
5. Scope

# import this

Write a function sleep_in(weekday, vacation) that tells us True/False can we sleep in. The parameter weekday is True if it is a weekday, and the parameter vacation is True if we are on vacation. We sleep in if it is not a weekday or we're on vacation. Return True if we sleep in.

sleep_in(False, False) → True
sleep_in(True, False) → False
sleep_in(False, True) → True

Given a string, we'll say that the front is the first 3 chars of the string. If the string length is less than 3, the front is whatever is there. Return a new string which is 3 copies of the front

front3('Java') → 'JavJavJav'
front3('Chocolate') → 'ChoChoCho'
front3('abc') → 'abcabcabc'
front3('q') → ‘qqq’

# Lists

Initialize an empty list:

```
>>> empty_list = []
>>> empty_list2 = list()
```

Initialize a list with elements:

```
>>> numbers = [1,2,3]
>>> mixed_list = [‘cat’, 3, [1,2,3]]
>>> 5_list = [None]*5
```

List length:

```
>>> mixed_list = [‘cat’, 3, [1,2,3]]
>>> len(mixed_list)
 3
```

['A', 'B', 'C', 'D', 'E']

Adding to a list:

```
>>> numbers = [1,2,3]
>>> numbers.append(4)
>>> numbers
 [1,2,3,4]
```

Joining lists:

```
>>> one_to_three = [1,2,3]
>>> four_to_six = [4,5,6]
>>> one_to_three + four_to_six
 [1,2,3,4,5,6]

>>> one_to_three.extend(four_to_six)
>>> one_to_three
 [1,2,3,4,5,6]
```

# SIPB: Programming in Python, IAP 2013

Changing an element:

```
>>> numbers = [1,2,3]
>>> numbers[0] = 100
>>> numbers
 [100,2,3]
```

Slicing:

```
>>> a = [‘Hello’, ‘World’, ‘!’]
>>> a[0:2]
 [‘Hello’, ‘World’]
```

Given an array of integers, return a new array length 2 containing the first and last elements from the original array. The original array will be length 1 or more.

make_ends([1, 2, 3]) → [1, 3]
make_ends([1, 2, 3, 4]) → [1, 4]
make_ends([7, 4, 6, 2]) → [7, 2]

# List Comprehension

# SIPB: Programming in Python, IAP 2013

List comprehension:

```
>>> [x**2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List comprehension with conditionals:

```
>>> coord = [12, -2, 5, -71, 39, 13, 4]
>>> [abs(x) for x in coord if x > 10 and x < 70]
 [12, 39, 13]
```

# Dictionaries

# SIPB: Programming in Python, IAP 2013

Initialize empty dictionary:

```
>>> empty_dict = {}
>>> empty_dict = dict()
```

Initialize dictionary with elements:

```
>>> dictionary = {1: 'The number 1', 'hello': 'A
common english greeting', 'the answer': 42}
```

Adding to a dictionary:

```
>>> d['key'] = 'value’
>>> d[variable] = other_value
```

# SIPB: Programming in Python, IAP 2013

Check if a key exists, then access dictionary element if it does:

```
>>> if foo in d:
            print d[foo]

```*If you try to access an element that isn't there, it will give you an error!*```

Joining dictionaries:

```
>>> dict1 = {‘Luke’: 1}
>>> dict2 = {‘Nathan’: 2}
>>> dict1.update(dict2)
>>> dict1
 {'Luke': 1, 'Nathan': 2}
```

# Other useful dictionary methods:

- .iteritems( )
- .iterkeys( )
- .keys( )
- .values( )
- .has_key( )

# Classes

# SIPB: Programming in Python, IAP 2013

Define a class:

```
class MyFirstClass:
    <statement-1>
        .
        .
    <statement-N>
```

Instantiate a class:

```
>>> a = MyFirstClass()
```

Classes can have attributes and methods:

```
class MyFirstClass:
    """A simple example class"""
    i = 12345
    def f(self):
        return 'hello world’
```

Initialize a class with arguments:

```
def MyFirstClass:
    def __init__(self, arg1, arg2, ...)
        self.arg1 = arg1
        self.arg2 = arg2
```

Class methods can then access those arguments:

```
    <... other methods ...>

    def change_color(self):
        if self.color == ‘RED’:
            self.color = ‘WHITE’
        else:
            self.color = ‘BLUE’

    <... more methods ...>
```

# Recursion

# SIPB: Programming in Python, IAP 2013

Recursion is when a function calls itself.

Recursion is usually used to solve complex problems, that can be broken down into smaller, identical problems.

Problems that can be solved with recursion, most likely can be solved with loops.

```
def print-x-n-times(x, n):
    # always have a base case!
    if (n <= 0):
     return
  print x
  print-x-n-times(x, n - 1)
```

Given a value N, return the Nth Fibonacci number.

Fibonacci numbers are computed by adding the previous two Fibonacci values.

1, 1, 2, 3, 5, 8, 13, 21, 34, …

# Scope

# Challenge Problem 1:
# http://tinyurl.com/aky9o5e

# Challenge Problem 2:
# Using a recursion, reverse a string
# ☺