
# MIT
# Global Startup Labs

http://aiti.mit.edu

Indonesia Summer 2013
Meetup 08 – More on Python

MISTI

Google

# Today‘s Meetup

- Short Recap of Yesterday
- More on Functions
- More on Datastructures
- Object Oriented Programming
- Today‘s Assignment

# Practice Examples

1.
```
x = 'Strawberry Juice'
y = x[-3:]*2 + x[5:10]
print y
```

2.
```
listex=[12, True, 'String']
if listex[1]:
    listex.append(len(listex)) #append() adds an element to the end of a list
print listex
```

3.
```
i=2
while i<10:
    i=i+1
    if i%2==0:
        continue
    print i
```

4.
```
for u in range(2, 12, 3):
    v=u**2
    print v
```

5.
```
listex2 = [x * y for x in [1, 3] for y in [2, 5] if x < y]
print listex2
```

# More on Functions

# Functions

- A function is a sequence of python statements that operates on predefined input parameters and can be called by a prespecified name

Function definition

Calling the function

```
def functionname(parameter1, parameter2):
    body


functionname(100, 'Hello')
```

# Functions

- Function arguments are local to function
- Program starts executing at first point after function definitions –> make sure to define or import functions BEFORE you use them
- Return statement ends function immediately

```
def convert_to_fahrenheit(c):
    tempC = 21  # this will NOT change
                # tempC outside the function!
    f = ((9.0 / 5.0) * c) + 32.0
    return f
    print("Hey") #after return -> will not print

tempC = 35
tempF = convert_to_fahrenheit(tempC)
print(tempF)
```

# Functions with variable arguments

Example: range() function can be called with one two or three arguments:

- Range(4) -> [0, 1, 2, 3]
- Range(5, 8) -> [5, 6, 7]
- Range(2, 14, 3) ->[2, 5, 8, 11]

How do we achieve this?

# Default arguments

- Use default argument values
  - They are used if the user does not specify a different value
- Call the function using positional or keyword arguments

```
def addorinc(x, y=1, name='Markus'):
    print name, ", you're awesome!"
    print x, '+', y, '=', x+y

addorinc(5)          #we only need to specify one value x=5

addorinc(5, 4)       #here we define the optional y=4

addorinc(2, 3, 'Bule')  #all three variables are defined using
                        #positional arguments

addorinc(3, name='wow') #using a keyword argument to define name

addorinc(x=4, y=2, name='whatever') #using three keyword arguments
```

# Functions example

```
def sentence(name, gender='he', prop='best', type='Bule'):
    print name, "is the", prop, "teacher ever,", gender,
    print 'is my favorite', type

sentence('Markus')

sentence('Nicole', 'she', type='Business Dude')

sentence('Lynn', name='Yu', gender='she')

sentence('Markus', prop='worst', 'white')
```

# More on datastructures

# More on sequence datatypes

Strings
- Elements can only be single characters, immutable

Lists
- Elements can be of any type, mutable

Tuples
- Elements can be of any type, immutable

Sets
- Unordered elements, no duplicates

Dictionaries
- Set of key-value pairs

# Sequence datatypes

- Tuples: use ()

```
>>> tuple1=(1, 12, 'hey')
>>> tuple[0]=2

Traceback (most recent call last):
  File "<pyshell#82>", line 1, in <module>
    tuple[0]=2
TypeError: 'type' object does not support item assignment
>>>
```

- Sets: use set() function

```
>>> set('alabama')
set(['a', 'b', 'm', 'l'])
>>> set([1, 2, 3, 3, 3, 5, 4, 2, 4])
set([1, 2, 3, 4, 5])
>>>
```

- Dictionaries: use {}

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['Markus'] = 1111
>>> tel
{'sape': 4139, 'jack': 4098, 'Markus': 1111}
>>> tel['jack']
4098
>>> 'Markus' in tel
True
>>>
```

# Sequence examples

Using Sets

Using Dictionaries

```
a = set('panama')
b = set('emma')

setc = a - b

setd = a & b


tel = {'jack': 4098, 'sape': 4139}

dic = {x: x**2 for x in (2, 4, 6)}
```

# Object Oriented Programming

# History of OOP

- Objects weren't always supported by programming languages
- Idea first originated at MIT in the 1960s and was officially incorporated in a few languages in the same decade
- OOP (Object Oriented Programming) has´now become a core feature of nearly all languages

# Using objects

We have already encountered many objects

For example: List object

```
>>> obj1 = [12, 2.32, 'Hello']
>>> obj1 = list([12, 2.32, 'Hello'])
>>> obj1
[12, 2.32, 'Hello']
>>> obj1.append(100)
>>> obj1
[12, 2.32, 'Hello', 100]
>>>
```

Creating a list object:

- Define the object class
- Pass on arguments

After we created an object, we can call methods specific to that class

# Creating a new class

```
class ExampleClass:

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

    def __str__(self):
        return ('hello world')

    # other basic customizations

    def addex(self):
        z=self.arg1+self.arg2
        print z

    def hw(self):
        print 'Hello World!'

    # other methods that you want to define
```

Define the name of the class

Basic customizations of class:

- __init__ -> called when class is first created
- __str__ -> called when print command on class
- Many more

Any functions you want to define: can be called as method objects later on

# Creating a new object

```
class ExampleClass:

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

    def __str__(self):
        return ('hello world')

    # other basic customizations

    def addex(self):
        z=self.arg1+self.arg2
        print z

    def hw(self):
        print 'Hello World!'

    # other methods that you wan
```

```
>>> var = ExampleClass(12, 21)
>>> var.arg1
12
>>> var.arg2
21
>>> print var
hello world
>>> var.addex()
33
>>> var.hw()
Hello World!
>>>
```

Create a new object of type ExampleClass

__str__ is executed

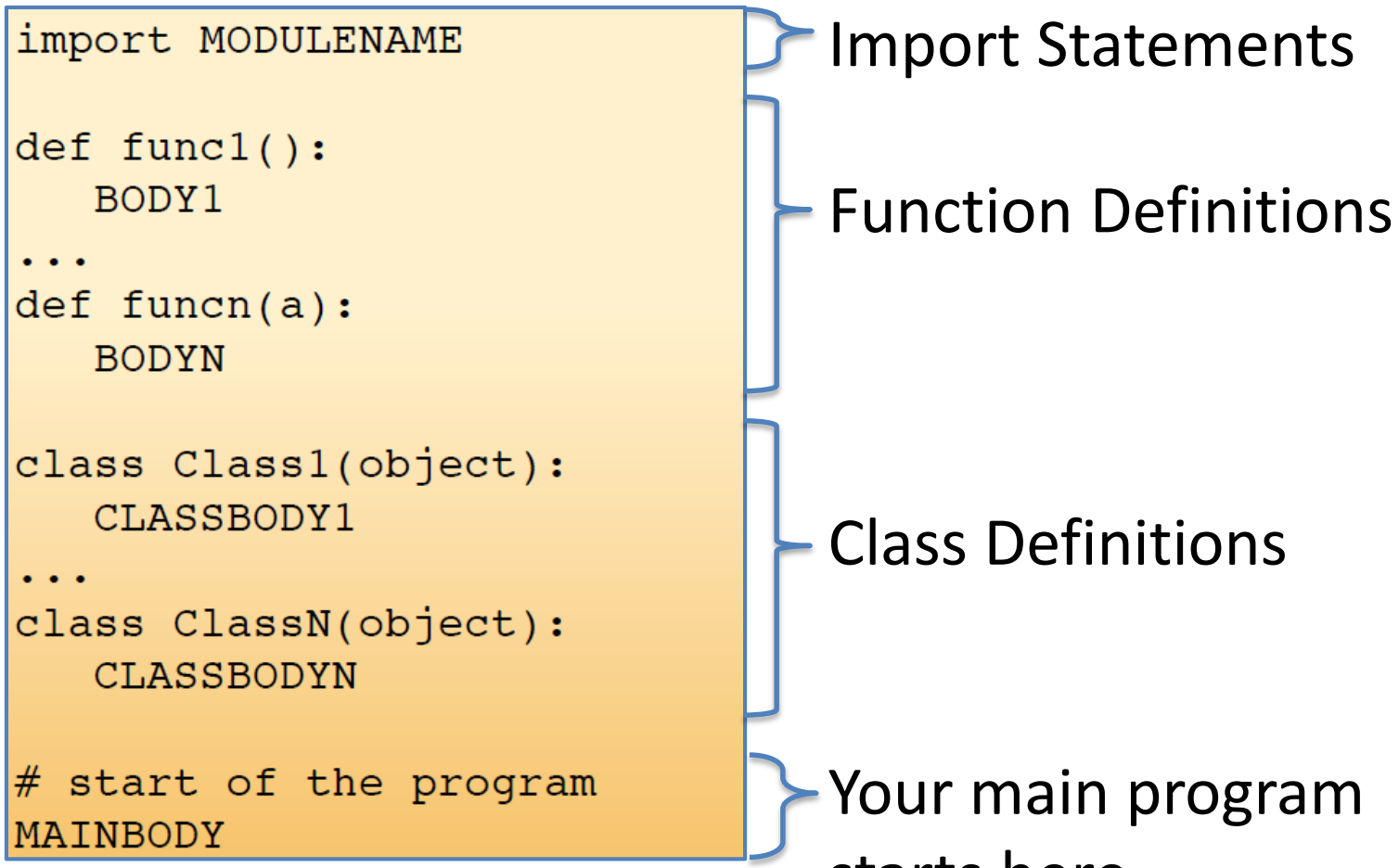
Method object var.addex() is created

Method object var.hw() is created

# Class Example

```
class example:

    def __init__(self, x):
        self.x = x

    def __str__(self):
        s='This is a wonderful number'
        return(s)

    def doublex(self):
        s=self.x*2
        print(s)

    def addx(self, y):
        return self.x + y


x = example(4)

x.doublex()

y = example(6)

print y

y.x

x.addx(y.x)
```

# Program organization



# Today‘s Assignment

# Today‘s Assignment

Lab 5: Python classes

- Setting up an Address Book

Get started on your project! Create your design documents

# Design Documents

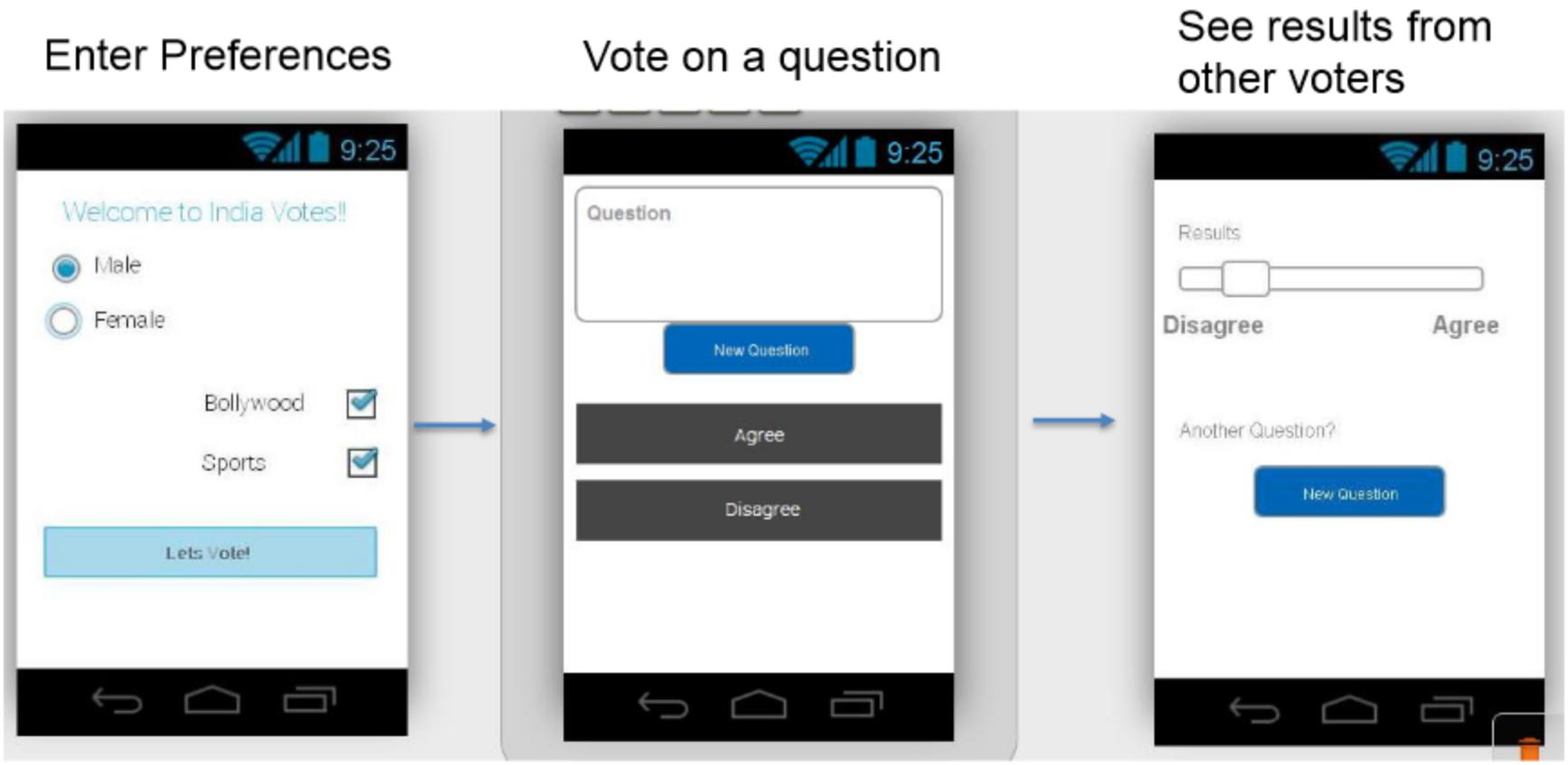
Three slides to give a concise overview of your app‘s outline. Include:

- Functional Requirements
- UI: Activities, buttons, processes
- A labeled information flow

# Design Documents Examples: India Votes (Requirements)

- Description: India votes is an app that allows users to vote on various current pop culture topics for fun
- The user must be able to login and set preferences for questions, and age group
- The app must generate questions based on the user's preferences and send the voting results to a central server
- The app must display the current voting results to the user and allow the user to respond to further questions if he/she chooses to do so

# Design Document Examples: India Votes (GUI)



# Design Document Examples: India Votes (Information Flow)