

# Introduction to Computer Programming

## COMS 127

Hebi Li

# Outline

- 1 Course Information
- 2 Survey
- 3 General intro to Programs
- 4 History of PL
- 5 Trends of PL
- 6 Overview at Python: a language perspective
- 7 Python Examples

# Team

- COMS 127: Introduction to Computer Programming
- Instructor: Prof. Forrest Bao



# Team

- COMS 127: Introduction to Computer Programming
- Instructor: Prof. Forrest Bao



- Hebi Li (me): random person, 1st week lecture

# Team

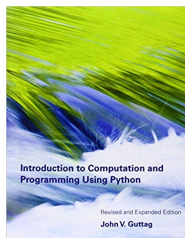
- COMS 127: Introduction to Computer Programming
- Instructor: Prof. Forrest Bao



- Hebi Li (me): random person, 1st week lecture
- TAs: TBD

# Syllabus

- Meeting times: Monday, Wednesday, Friday 2:10-3:00 PM
- Location: Food Science 2432
- Textbook: Guttag, John V. *Introduction to computation and programming using Python*. Mit Press, 2013.



# Plan for this week

## Monday: Introduction to programming

- Intro to Programming
- Programming history & trends
- Introduction to python

## Wednesday: Setup python enviroment

Bring your laptop!

- Editors
- IDEs
- Online interpreters: Jupyter notebooks

## Friday: expressions and variables

# TOC

- 1 Course Information
- 2 Survey**
- 3 General intro to Programs
- 4 History of PL
- 5 Trends of PL
- 6 Overview at Python: a language perspective
- 7 Python Examples



# Have you ever writing any programs?

If so, which is your favorite, and why?

# Have you ever writing any programs?

If so, which is your favorite, and why?

"Main stream"

- ☐ C/C++
- ☐ Java
- ☐ Javascript
- ☐ python

# Have you ever writing any programs?

If so, which is your favorite, and why?

## "Main stream"

- ☐ C/C++
- ☐ Java
- ☐ Javascript
- ☐ python

## Other options


- ☐ scripting languages: perl, ruby, shell script
- ☐ Lisp dialect: common lisp, clojure, scheme
- ☐ Domain specific: R
- ☐ shining & new languages: rust, go, julia

# What operating system do you use

- ☐ Linux
- ☐ Windows
- ☐ Mac OS

# What editor do you use?

☐ notepad 

☐ Sublime text 


☐ Atom 

☐ Eclipse 

☐ VS Code 

☐ Emacs 

☐ Vim 

☐ Online editors (e.g. jupyter notebook) 

# TOC

- 1 Course Information
- 2 Survey
- 3 General intro to Programs**
- 4 History of PL
- 5 Trends of PL
- 6 Overview at Python: a language perspective
- 7 Python Examples

# What is a program?

## wikipedia definition

*A computer program is a **collection of instructions** that performs a specific task when executed by a computer.*

...

*A computer program is usually written by a computer programmer in a **programming language**.*

...

*From the program in its human-readable form of source code, a **compiler** or assembler can derive machine code. ... Alternatively, a computer program may be executed with the aid of an **interpreter**.*

# Hello World!

Hello world:

```
1  print ("Hello World")
```



# Hello World!

Hello world:

```
1 print ("Hello World")
```

print is a library call. You can print anything out in python:

```
1 print(1)
2 print(1+2)
3 1+2
```

# Major components of computer programs

## Basic ingredients

- variables
- expressions
- statements

# Major components of computer programs

## Basic ingredients

- variables
- expressions
- statements

## Control flow

- conditional: if .. else ..
- loop: for-loop

# Major components of computer programs

## Basic ingredients

- variables
- expressions
- statements

## Control flow

- conditional: if .. else ..
- loop: for-loop

## data structures

- list
- hash tables (dictionaries)

# Major components of computer programs

## Basic ingredients

- variables
- expressions
- statements

## Abstraction

- Functions
- Classes

## Control flow

- conditional: if .. else ..
- loop: for-loop

## data structures

- list
- hash tables (dictionaries)

# Major components of computer programs

## Basic ingredients

- variables
- expressions
- statements

## Abstraction

- Functions
- Classes

## Control flow

- conditional: if .. else ..
- loop: for-loop

## project organization

- modules & files

## data structures

- list
- hash tables (dictionaries)

# Major components of computer programs

## Basic ingredients

- variables
- expressions
- statements

## Abstraction

- Functions
- Classes

## Control flow

- conditional: if .. else ..
- loop: for-loop

## project organization

- modules & files

## data structures

- list
- hash tables (dictionaries)

## libraries

- standard libraries
- 3rd-party libraries

# Basic ingredients

- variables
- expressions
- statements

---

```
1  pi = 3.14                # variable: pi
2  r = 10
3  pi * r * r               # expression, evaluates to 314
4  print(pi * r * r)
5  area = pi * r * r        # assignment statement
6  print(area)              # print statement
```

---



# Control flow

- conditional: if .. else ..
- loop: for-loop

# Control flow

- conditional: if .. else ..
- loop: for-loop

```
_____ conditional _____  
1  a = -3  
2  if a > 0:  
3      print('a is positive')  
4  elif a < 0:  
5      print('a is negative')  
6  else:  
7      print('a = 0')
```

# Control flow

- conditional: if .. else ..
- loop: for-loop

```

_____ conditional _____
1  a = -3
2  if a > 0:
3      print('a is positive')
4  elif a < 0:
5      print('a is negative')
6  else:
7      print('a = 0')
_____

```

```

_____ loop _____
1  for i in range(2, 10):
2      print(i)
3
4  for i in range(2, 10):
5      if is_prime(i):
6          print(i)
_____

```

# data structures

- list
- hash tables (dictionaries)

# data structures

- list
- hash tables (dictionaries)

```

1  list
   l = [10, 8, 9, 1, 5, 6]
2  print(l[0])    # => 10
3  print(l[1])    # => 8
4  print(l[2:4])  # => [9, 1]
5  print(len(l))  # => 6
6  print(sorted(l))
7  # => [1, 5, 6, 8, 9, 10]

```

# data structures

- list
- hash tables (dictionaries)

```

_____ list _____
1  l = [10, 8, 9, 1, 5, 6]
2  print(l[0])    # => 10
3  print(l[1])    # => 8
4  print(l[2:4])  # => [9,1]
5  print(len(l))  # => 6
6  print(sorted(l))
7  # => [1, 5, 6, 8, 9, 10]
_____

```

```

_____ dict _____
1  d = dict()
2  d["hello"] = "world"
3  d[2] = "two"
4  d["five"] = 5
5  print(d["hello"])
6  # => "world"
_____

```

# Functions

```
_____ statements _____  
1  pi = 3.14  
2  r = 10  
3  area = pi * r * r  
4  print(area)  
5  
6  r2 = 11  
7  area2 = pi * r2 * r2  
8  r3 = 12  
9  area3 = pi * r3 * r3  
_____
```

# Functions

```
_____ statements _____  
1  pi = 3.14  
2  r = 10  
3  area = pi * r * r  
4  print(area)  
5  
6  r2 = 11  
7  area2 = pi * r2 * r2  
8  r3 = 12  
9  area3 = pi * r3 * r3  
_____
```

- For **code reuse** and abstraction.



# Functions

```

_____ statements _____
1  pi = 3.14
2  r = 10
3  area = pi * r * r
4  print(area)
5
6  r2 = 11
7  area2 = pi * r2 * r2
8  r3 = 12
9  area3 = pi * r3 * r3
_____

```

- For **code reuse** and abstraction.

```

_____ function _____
1  def circle_area(r):
2      pi = 3.14
3      return pi * r * r
4
5  for r in range(10):
6      print(circle_area(r))
_____

```

# Classes: object-oriented programming

Class consists of

- **fields**: class local variables
- **methods**: functions to operate on the fields

# Classes: object-oriented programming

Class consists of

- **fields**: class local variables
- **methods**: functions to operate on the fields

---

```
1 class Shape():  
2     def area(self):  
3         raise NotImplementedError
```

---

# Classes: object-oriented programming

Class consists of

- **fields**: class local variables
- **methods**: functions to operate on the fields

---

```
1 class Shape():
2     def area(self):
3         raise NotImplementedError
```

---

---

```
1 class Circle(Shape):
2     def __init__(self, r):
3         self.r = r
4     def area(self):
5         return (3.14 * self.r
6                 * self.r)
```

---

# Classes: object-oriented programming

Class consists of

- **fields**: class local variables
- **methods**: functions to operate on the fields

---

```

1  class Shape():
2      def area(self):
3          raise NotImplementedError

```

---



---

```

1  class Circle(Shape):
2      def __init__(self, r):
3          self.r = r
4      def area(self):
5          return (3.14 * self.r
6                  * self.r)

```

---



---

```

1  class Rectangle(Shape):
2      def __init__(self, w, h):
3          self.w = w
4          self.h = h
5      def area(self):
6          return self.w * self.h

```

---

# project organization

- modules & files

```
_____ lib.py _____  
1  def area(r):  
2      pi = 3.14  
3      return pi * r * r  
4  def perimeter(r):  
5      pi = 3.14  
6      return 2 * r * pi  
_____
```

# project organization

- modules & files

```

_____ lib.py _____
1  def area(r):
2      pi = 3.14
3      return pi * r * r
4  def perimeter(r):
5      pi = 3.14
6      return 2 * r * pi
_____

```

```

_____ use.py _____
1  from lib import area
2  from lib import perimeter
3  r = 10
4  print('Area: ', area(r))
5  print('Perimeter: ',
6        perimeter(r))
_____

```

# Python standard library <sup>1</sup>

- numerical: `math`, `random`
- File IO: `os`, `io`
- dataset: `pickle`, `sqlite3`
- GUI: `tkinter`
- multi-thread: `threading`, `subprocess`
- network: `socket`, `asyncio`
- web: `html`, `xml`, `http`

---

<sup>1</sup><https://docs.python.org/3/library/>



# Recap: program components

## Basic ingredients

- variables
- expressions
- statements

## Control flow

- conditional: if .. else ..
- loop: for-loop

## data structures

- list
- hash tables (dictionaries)

## Abstraction

- Functions
- Classes

## project organization

- modules & files

## libraries

- standard libraries
- 3rd-party libraries


# Python documents

- python language reference:  
<https://docs.python.org/3/reference/>
- python standard library: <https://docs.python.org/3/library/>


# TOC

- 1 Course Information
- 2 Survey
- 3 General intro to Programs
- 4 History of PL**
- 5 Trends of PL
- 6 Overview at Python: a language perspective
- 7 Python Examples

# Early days (50s-60s)

- 1957: FORTRAN 
- 1958: Lisp-1.5: code is data
- 1958 ALGOL
  - ALGOL 58
  - ALGOL 60
  - ALGOL 68
- 1959 COBOL
- 1964 BASIC
- 1966 APL

## Early days (50s-60s)




- 1957: FORTRAN 
- 1958: Lisp-1.5: code is data
- 1958 ALGOL
  - ALGOL 58
  - ALGOL 60
  - ALGOL 68
- 1959 COBOL
- 1964 BASIC
- 1966 APL

*Lisp is worth learning for the profound enlightenment experience you will have when you finally get it; that experience will make you a better programmer for the rest of your days, even if you never actually use Lisp itself a lot.*  
– Eric Raymond, in *How to Become a Hacker*




# 1970s: Establishing fundamental paradigms

- 1970: Pascal, in ALGOL family
- 1972: C
  - 1973: Unix V4 rewritten in C
- 1972: Smalltalk
- 1972: Prolog
- 1973: ML, "Lisp with types"
- 1975 Scheme: cleanly designed Lisp
  - proper lexical scoping
  - hygienic macros
- 1978 SQL

# 1980s

- 1980: C++ 
  - add object-oriented to C
- 1980: Ada
- 1984: Common Lisp
- 1984: Matlab
- 1986: Erlang 
- 1986: Labview
  - graphical programming
- 1986: Perl 
- 1988: Tcl

## 1980s








- 1980: C++ 
  - add object-oriented to C
- 1980: Ada
- 1984: Common Lisp
- 1984: Matlab
- 1986: Erlang 
- 1986: Labview
  - graphical programming
- 1986: Perl 
- 1988: Tcl

Philip Greenspun:








*Any sufficiently complicated C or Fortran program contains an ad hoc informally-specified bug-ridden slow implementation of half of Common Lisp.*



# 1990s: the Internet age, developer productivity








- 1990: Haskell 
  - pure-functional language
  - Monad
- 1991: Python 
  - scripting made easier
- 1991: Visual Basic
- 1993: Lua 
- 1993: R 
- 1995: Ruby 
- 1995: Java 
  - most popular for 2 decades
- 1995: PHP 
- 1995: Javascript

# 1990s: the Internet age, developer productivity

- 1990: Haskell 
  - pure-functional language
  - Monad
- 1991: Python 
  - scripting made easier
- 1991: Visual Basic
- 1993: Lua 
- 1993: R 
- 1995: Ruby 
- 1995: Java 
  - most popular for 2 decades
- 1995: PHP 
- 1995: Javascript

*Atwood's Law: any application that can be written in JavaScript, will eventually be written in JavaScript.*

## 2000-2010

- 2001: C# 
- 2001: D 
- 2002: **scratch**  Demo:  
<https://scratch.mit.edu>
- 2003: scala 
- 2005: F# 
- 2007: Clojure 
- 2009: Go 

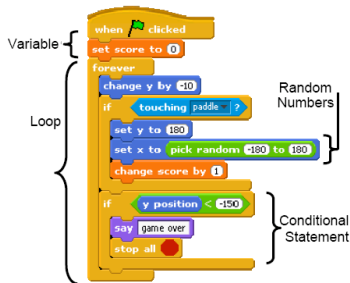


Figure: a Scratch program

## 2010+





- 2010: Rust: memory safety 
- 2011: Dart: cross platform (Apple, Android, Web) 
- 2011: Kotlin: can we fix Java? 
- 2011: Elixir 
- 2012: Julia: machine learning, the new R 
- 2012: Typescript: Javascript with static typing 
- 2014: Swift 

## 2010+

- 2010: Rust: memory safety 
- 2011: Dart: cross platform (Apple, Android, Web) 
- 2011: Kotlin: can we fix Java? 
- 2011: Elixir 
- 2012: Julia: machine learning, the new R 
- 2012: Typescript: Javascript with static typing 
- 2014: Swift 

What will be the new python in 5 years?

# Lisp family

- 1958: lisp 1.5
  - 1975: scheme
    - 1995: R5RS
    - 2005: R6RS
    - 2010: R7RS
  - 1984: common lisp, 1994 ANSI CL with CLOS
  - 1985: Emacs Lisp 
  - 1993: GNU guile 
  - 1995: racket 
  - 2007: Clojure 

# TOC

- 1 Course Information
- 2 Survey
- 3 General intro to Programs
- 4 History of PL
- 5 Trends of PL**
- 6 Overview at Python: a language perspective
- 7 Python Examples

# TIOBE Index <sup>2</sup>

Aug 2019	Aug 2018	Programming Language	Ratings
1	1	Java	16.028%
2	2	C	15.154%
3	4	Python	10.020%
4	3	C++	6.057%
5	6	C#	3.842%
6	5	Visual Basic .NET	3.695%
7	8	JavaScript	2.258%
8	7	PHP	2.075%
9	14	Objective-C	1.690%
10	9	SQL	1.625%
24		Dart	0.715%
28		Rust	0.450%
29		Scratch	0.448%
34		Lisp	0.362%
39		Julia	0.279%
49		Haskell	0.174%

<sup>2</sup>Data from search engine. <https://www.tiobe.com/tiobe-index/>

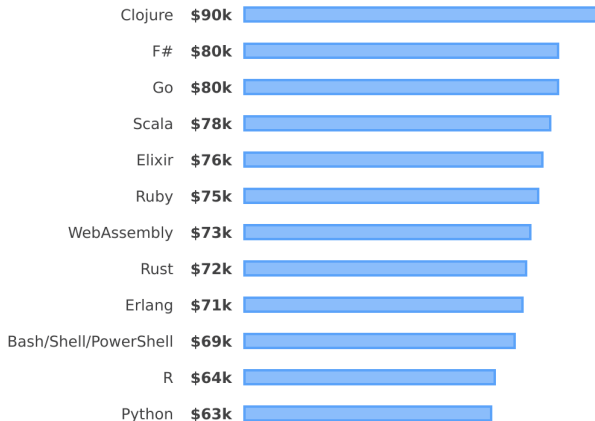


# TIOBE Index history

PL	2019	2014	2009	2004	1999	1994	1989
Java	1	2	1	1	14	-	-
C	2	1	2	2	1	1	1
Python	3	7	5	7	24	21	-
C++	4	4	3	3	2	2	2
VB .NET	5	9	-	-	-	-	-
C#	6	5	6	6	19	-	-
JavaScript	7	8	8	8	16	-	-
PHP	8	6	4	5	-	-	-
SQL	9	-	-	89	-	-	-
Objective-C	10	3	31	38	-	-	-
Perl	16	11	7	4	3	10	22
Lisp	32	13	19	13	12	5	3
Pascal	220	16	14	88	6	3	20

# Stack Overflow Developer Survey 2019 <sup>3</sup>

What Languages Are Associated with the Highest Salaries Worldwide?



<sup>3</sup>[https:](https://insights.stackoverflow.com/survey/2019#top-paying-technologies)

[//insights.stackoverflow.com/survey/2019#top-paying-technologies](https://insights.stackoverflow.com/survey/2019#top-paying-technologies)

# Uncle Bob Aug 22, 2019 <sup>4</sup>

## The Clean Code Blog

by Robert C. Martin (Uncle Bob)

atom/rss feed

- Why Clojure?  
08-22-2019
- Why won't it...  
07-22-2019
- Classes vs. Data Structures  
06-16-2019
- Types and Tests  
06-08-2019
- 737 Max 8  
05-18-2019
- FP vs. OO List Processing  
12-17-2018
- We, The Unoffended  
12-16-2018
- SJWJS  
12-14-2018
- The Tragedy of

## Why Clojure?

22 August 2019

I've programmed systems in many different languages; from assembler to Java. I've written programs in binary machine language. I've written applications in Fortran, COBOL, PL/1, C, Pascal, C++, Java, Lua, Smalltalk, Logo, and dozens of other languages. I've used statically typed languages, with lots of type inference. I've used typeless languages. I've used dynamically typed languages. I've used stack based languages like Forth, and logic based languages like Prolog.

Over the last 5 decades, I've used a LOT of different languages.

And I've come to a conclusion.

My favorite language of all, the language that I think will outlast all the others, the language that I believe will eventually become the standard language that all programmers use...

...is Lisp.

<sup>4</sup><http://blog.cleancoder.com/uncle-bob/2019/08/22/WhyClojure.html>

racket demo?? <sup>5</sup>


---

```

1  #lang racket
2  (require 2htdp/planetcute)
3  (require 2htdp/image)
4  (define (stack imgs)
5    (cond
6      [(empty? (rest imgs)) (first imgs)]
7      [else (overlay/xy (first imgs)
8                          0 40
9                          (stack (rest imgs)))]))
10 (beside/align
11   "bottom"
12   (stack (list wall-block-tall stone-block))
13   (stack (list character-cat-girl
14             stone-block stone-block
15             stone-block stone-block))
16   water-block
17   (stack (list grass-block dirt-block))
18   (stack (list grass-block dirt-block dirt-block)))

```

---

<sup>5</sup><https://docs.racket-lang.org/teachpack/2htdpPlanetCute/Images.html>

# TOC

- 1 Course Information
- 2 Survey
- 3 General intro to Programs
- 4 History of PL
- 5 Trends of PL
- 6 Overview at Python: a language perspective**
- 7 Python Examples

# Python

Table: Python language feature

Features	Option 1	Option 2
Compilation model	interpreted	compiled
Typing checking	strong	weak
Variable typing	static	dynamic
Scoping	lexical	dynamic
Paradigm	procedural	functional
Pure-functional	pure	impure
Memory model	manual	garbage collected
Object-Oriented	Yes	No

# Compiler vs. Interpreter

## Compiler

source code **compiled to object code** then runs

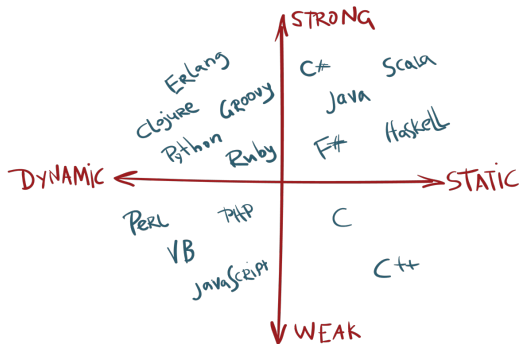
- check errors during compilation
- whole-program optimization, thus **run fast**

## Interpreter

source code, directly run

- **Interactive**
- Incremental development, thus **develop fast**

## strong/weak &amp; static/dynamic Typing



- Strong/Weak typing: Whether type safety is enforced on **values**.
- Static/Dynamic typing: Whether **variables** are typed.



# strong typing & weak typing

Whether type safety is enforced on **values**.

Strong typing: python

```
1 1 + "hello"
```

# strong typing & weak typing

Whether type safety is enforced on **values**.

Strong typing: python

```
1 1 + "hello"
```

***TypeError:** unsupported operand type(s) for +: 'int' and 'str'*

# strong typing & weak typing

Whether type safety is enforced on **values**.

Strong typing: python

```
1 1 + "hello"
```

***TypeError:** unsupported operand type(s) for +: 'int' and 'str'*

Weak typing: Javascript

```
1 1 + "hello"
```

# strong typing & weak typing

Whether type safety is enforced on **values**.

## Strong typing: python

```
1 1 + "hello"
```

***TypeError:** unsupported operand type(s) for +: 'int' and 'str'*

## Weak typing: Javascript

```
1 1 + "hello"
```

*"1hello"*

# static & dynamic typing

Whether **variables** are typed.

static typing: C

```
1  int var;  
2  var = 8;  
3  var = "hello";  // compile error
```

# static & dynamic typing

Whether **variables** are typed.

static typing: C

```
1  int var;  
2  var = 8;  
3  var = "hello";  // compile error
```

dynamic typing: python

```
1  var = 8  
2  var = "hello"
```

# memory safety & management

- manual memory management
- automatic memory management: gabbage collection
- ownership (Rust)

# memory safety & management

- manual memory management
- automatic memory management: garbage collection
- ownership (Rust)

## Manual memory management: C

```
1  int *var;  
2  var = (int*) malloc(sizeof(int));  
3  *var = 5;  
4  free(var);
```



# memory safety & management

- manual memory management
- automatic memory management: garbage collection
- ownership (Rust)

## Manual memory management: C

```
1  int *var;  
2  var = (int*) malloc(sizeof(int));  
3  *var = 5;  
4  free(var);
```

## Gabbage collected: python

```
1  l = [1,2,3]  
2  l.append(4)  
3  print(l)      # => [1,2,3,4]
```

# functional & procedural programming

functional:

- function is a **first-class citizen**
  - *first-class* means a function can appear anywhere value can occur
  - can be assigned to variable, pass to a function, or returned by a function
- function in *mathematical sense*: given input, compute output. There is **no side effect**.

---

```

1 var = 8
2 def bad_square(a):
3     global var
4     var = a
5     return a * a

```

---



---

```

1 print(var) # => 8
2 print(bad_square(3))
3 # => 9
4 print(var) # => 3

```

---

# TOC

- 1 Course Information
- 2 Survey
- 3 General intro to Programs
- 4 History of PL
- 5 Trends of PL
- 6 Overview at Python: a language perspective
- 7 Python Examples**

# Prime number

*A prime number (or a prime) is a natural number greater than 1 that cannot be formed by multiplying two smaller natural numbers.*

- composite number: 6 is not prime, because  $6 = 2 * 3$
- list of prime numbers: 2, 3, 5, 7, 11, 13, ...

---

```
1 def is_prime(n):
2     for i in range(2, round(n/2)+1):
3         if n % i is 0:
4             return False
5     return True
6
7 for i in range(2, 10):
8     if is_prime(i):
9         print(i)
```

---

# Fibonacci number

$$F_0 = 1$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} \quad \text{for } n > 1$$

Example: 1, 1, 2, 3, 5, 8, 13

# Fibonacci number

$$F_0 = 1$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} \quad \text{for } n > 1$$

Example: 1, 1, 2, 3, 5, 8, 13

---

```
1 def fib(n):  
2     if n is 0: return 1  
3     if n is 1: return 2  
4     return fib(n-1) + fib(n-2)  
5  
6 for i in range(10):  
7     print(fib(i))
```

---

# Training a neural network classifier

Online runtime: <https://colab.research.google.com>

Import libraries:

---

```
1 import keras
2 from keras.layers import Input, Dense, Flatten
3 from keras.models import Model
4 from keras.datasets import mnist
5 import random
6 import matplotlib.pyplot as plt
7 import numpy as np
```

---

# Training a neural network classifier

Setup model:

---

```
1 inputs = Input(shape=(28,28,))
2 x = Flatten()(inputs)
3 x = Dense(64, activation='relu')(x)
4 x = Dense(64, activation='relu')(x)
5 predictions = Dense(10, activation='softmax')(x)
6 model = Model(inputs, predictions)
7 model.compile(optimizer='rmsprop',
8               loss='categorical_crossentropy',
9               metrics=['accuracy'])
```

---



# Training a neural network classifier

Load MNIST data and train the model:

---

```
1 (x_train, y_train), (x_test, y_test) = mnist.load_data()
2 y_train = keras.utils.to_categorical(y_train, 10)
3 y_test = keras.utils.to_categorical(y_test, 10)
4
5 model.fit(x_train, y_train, epochs=10)
6 model.evaluate(x_test, y_test)
```

---

# Training a neural network classifier

## Visualization:

---

```
1 index = random.randint(0, 1000)
2 plt.imshow(x_train[index])
3 plt.savefig("image.jpg")
4 pred = np.argmax(
5     model.predict(
6         np.array([x_train[index]])) [0])
7 print("prediction: ", pred)
```

---