



## Kickstart Cheatsheets

### **A selection of printable, one-page cheatsheets.**

These were originally designed for Kickstart Coding, the affordable, inclusive, and intensive coding course teaching cutting-edge Python / Django and JavaScript / React web development in Oakland, CA. Learn more and enroll [here](#).

### **Download**

Click on one of the following to download. These are all 1 page long, and look great when printed. Consider printing to hang up near your computer as you code or learn a new programming language!

### **Modern HTML/CSS (2021)**

A cheat-sheet containing the most commonly used HTML tags and CSS selectors, and key terms in modern HTML5 & CSS3, including CSS Grid and pseudo-selectors.

## COMMON HTML TAGS

name attribute(s)  
 ↓  
 <p class="my-class"> ← opening  
 Some text ← contents  
 </p> ← closing

## Paragraph

<p>Paragraph of text</p>

## Styling

<em>Italic</em> and  
 <strong>bold</strong>

## Image



## Link (aka "anchor")

<a href="http://google.com">  
 Link to Google  
 </a>

## Headers

<h1>Huge</h1> <h2>Big</h2>  
 ... <h6>Small</h6>

## Freeform

<div></div> (block) and  
 <span></span> (inline)

## Semantic

<section>, <article>, and more

## Comment

<!-- ignored text -->

## BOX MODEL



Modern HTML (2021)

## CSS EXAMPLES

selector  
 ↓  
 { declaration block  
 font-size: 16px;  
 }  
 property value

## Text

p {  
 font-size: 16px;  
 text-align: center;  
 color: green;  
 font-family: "Arial";  
 font-weight: bold;  
 line-height: 20px;  
 }

## Sizing

div {  
 height: 30px;  
 width: 100%;  
 margin-left: 20px;  
 padding-top: 10px;  
 }

## Block styling

.highlighted-area {  
 display: block;  
 border: 1px solid black;  
 background: yellow;  
 border: 2px solid green;  
 border-radius: 10px;  
 }

## Transition (animates property)

.fading-link {  
 transition: color 1s,  
 padding-left 3s;  
 }  
 .fading-link:hover {  
 color: green;  
 padding-left: 20px;  
 }

## CSS DISPLAY

block fill available width, word-wrap  
 inline children (e.g. paragraph)  
 inline word-wrap (e.g. strong)  
 inline-block square block within word-  
 wrapped text (e.g. emoji or icon)  
 flex line up children, control spacing  
 grid position children in grid

kickstartcoding.com

## HTML & CSS (2021)

### CSS SELECTORS

Tag div { color: blue; }  
 Class .class-name { color: blue; }  
 ID #id\_name { color: blue; }

### Containment (match children)

/\* Match all divs in #a \*/  
 #a div { color: blue; }  
 /\* Divs immediately in #b \*/  
 #b > div { color: blue; }  
 /\* Wildcard: anything in #c \*/  
 #c \* { color: blue; }

### Pseudo-elements (add content)

h1::before { content: "~"; }

### Pseudo-classes (match by position)

/\* First p margin \*/  
 p:nth-child(1) { margin: 10px; }  
 /\* Striped table rows \*/  
 tr:nth-child(odd) { color: gray; }

### GRID CONTAINER

grid-template-columns grid-template-rows  
 1fr 1fr 1fr 1fr 1fr 1fr  
 50px auto 50px 50px auto 90px  
 20% 20% auto 100px auto 20%

.container-element {  
 display: grid;  
 grid-template-columns: 1fr 1fr 1fr;  
 grid-template-rows: 50px auto 50px;  
 }

### GRID CHILD POSITIONING

grid-column grid-row  
 2 / 3  
 2 / span 2  
 1 / span 2  
 1 / 3  
 1 / span 2

.child-element {  
 grid-column: 1 / span 2;  
 grid-row: 1 / span 2;  
 }

A cheatsheet by Kickstart Coding

## CLI: Bash & Git

Learning CLI and Git usage on macOS or Linux? This cheatsheet has the most commonly used commands in Bash and Git. It also contains more advanced usage, such as using `grep`, `find`, piping, redirection, job and process control, and interacting with the bash history.



## BASH BASICS

## Navigating

```
cd name_of_directory
cd .. # Go up one
cd ~ # Go to home
pwd # Where am I?
```

## Listing files

```
ls # List files
ls -a # See hidden
ls -l # See more info
ls -R # Recursive
```

## Moving and renaming

```
mv file.txt new_name.txt
mv file.txt ../new/place/
```

## Copying

```
cp file.txt file_backup.txt
cp -r directory/ backup/
```

## Deleting

```
rm file.txt
rm -r empty_directory/
rm -r full_directory/
```

## Creating

```
mkdir my_directory
touch empty_file.py
```

## Reading data from file

```
cat filename.txt
cat file1 file2 file3
```

## BASH TRICKS

## Auto complete Start typing then hit &lt;Tab&gt;. Hit twice for options.

## Redirecting output into file

```
ls -R > all_files.txt
cat a.html b.html > c.html
cat d.txt >> c.txt # Append
```

## Piping output Hook commands up

```
# Pipe output to "grep" filter
python start.py | grep "http"
node run.js | tail # Only end
```

## Wildcard expansions

```
rm *.jpg # Delete jpg files
rm ./**/*.jpg # ** matches dirs
```

## Running file as bash script

```
# Save a sequence of commands to
# file with "cat" or "tee" at top
bash script.sh # Always works
./script.sh # Works if executable
```

## BASH: VARIABLES, OUTPUT

## Setting and viewing variables

```
PLANET="world"
echo "Hello $PLANET"
env # Show ALL variables
```

## Run command with variable set

```
DEBUG=true apt start
```

## BASH: HISTORY

## History commands

```
cd - # go back a directory
history # view all commands
!! # last command you typed
sudo !! # sudo, but no sudo
```

## Shortcut: Last command &lt;Ctrl&gt;

```
<Ctrl>R then start typing
<Ctrl>R to cycle back,
<Enter> to run.
```

## BASH: PROCESS MANAGEMENT

## Multiple commands

```
c1 ; c2 # run c2 after c1
c1 && c2 # run c2 if c1 succeeds
c1 || c2 # run c2 if c1 fails
c1 & c2 # run both at once
```

## Job control

```
apt start & # run in bg
ps # show shell's processes
jobs # show bg processes
fg # foreground last process
<Ctrl>Z # pause; put in bg
# keep background process [!]  
disown !! # running forever
```

## Viewing all processes

```
ps -e # show all processes
ps -ejR # show process trees
ps -e | grep python # filter
```

## Killing processes

```
kill 4264 # kill process by PID
killall python # ... or by name
kill -9 4264 # -9 "force" kill
```

## GIT

## Starting (local) repo

```
git init
```

## Starting with repo from GitHub

```
# Using HTTP (prompt for pw)
git clone https://github.com/U/R
# Using SSH (requires setup)
git clone git@github.com:U/R.git
```

## Adding and committing

```
git add -A # "Stage" all
git commit -m "Fixed :)"
```

## Finding out status

```
git status
git log
```

## Learning about past

```
git log # Q to quit
git show f88bdcf
git diff f88bdcf master
git checkout f88bdcf
```

## Branch workflow

```
git branch my-stuff
git checkout my-stuff
# After you do some work...
git add -A
git commit -m "New logs :)"
git checkout master
git merge my-stuff
```

## Interacting with remotes

```
(e.g. GitHub, Heroku)
git remote -v # check remotes
git pull # get updates
# After you do some work...
git add -A
git commit -m "it works!"
git push # share updates
```

## BASH: SEARCHING

## find: Search by filename

```
# Using wildcards for search by
find . -name "*.py" # extension
find . -name views.py # Exact
find . -iname l1fo # Any case
# Find modified in last 7 days
find . -mtime 7 -iname l1fo
```

## grep: Search contents of files

```
# Search templates for "free"
grep -r free ./templates/
grep -lr free . # ...list name
grep -lr l1fo . # Ignore case
# Using Regular Expressions
grep -r "[a-zA-Z]*"
```

## Python

A cheatsheet with common tasks in Python. Designed for Python 3.x, but will mostly work with 2, also.

| TYPES  | BRANCHING   | FUNCTIONS  |
|--|---|--|
| <b>str, int, float</b><br><pre>a = "hello" # string count = 3 # integer pi = 3.14 # float</pre>  | <b>Basic if</b> Conditionally execute indent<br><pre>if cost &lt; 10:     print("impulse buy")</pre>  | <b>Return value w/ positional param</b><br><pre>def in_file(name):     path = "../src/" + name     return path + ".html" path = in_file("home") html = open(path).read()</pre>   |
| <b>list ordered collection</b><br><pre>a = ["a", "b", 3] a[0] # "a" a[1] # "b" a[-1] # "3" a[1:2] # ["b", 3]</pre>   | <b>Boolean operators</b> "and", "or"<br><pre>if age &gt; 17 and place == "UK":     print("can buy alcohol") if age &lt; 18 or a == "student":     print("can get discount")</pre>           | <b>Keyword parameters</b><br><pre>def greet(name="Jack"):     print("Hello", name) greet(name="Jill")</pre>  |
| <b>tuple same as list, but immutable</b><br><pre>a = ("a", "b", 3)</pre>   | <b>If-else</b><br><pre>if beer == "Darkwing":     print("IPA") elif beer == "Stonehenge":     print("Stout") else:     print("Unknown beer")</pre>  | <b>Variable length arguments</b><br><pre>def do_all(*args, **kwargs):     print(kwargs) # kwargs is dict     return sum(args) do_all(3, 5, b=3)</pre>  |
| <b>dict collection of keys and values</b><br><pre>a = {"test": 1, "b": "hello"} a["test"] # 1 a["b"] # "hello" del a["test"] # delete "test" a["c"] = 3 # add "c" to dict</pre>                  | <b>Pass placeholder that does nothing</b><br><pre>if cost &gt; 1.99:     pass # TODO: finish this</pre>   | <b>Comment</b> aka "dostring"<br><pre>def plural(word):     """     Return the plural of     an English word.     """     if word.endswith("s"):         return word + "es"     return word + "s" print("Many", plural("cat"))</pre> |
| <b>sets "keys-only dict", with operations</b><br><pre>a = {"a", 1, 4, "b"} b = {"a", "b"} print(a - b) # {1, 4}</pre>  | <b>ITERATION</b><br><b>While loop</b> Repeat indented code until condition is no longer true<br><pre>i = 2 while i &lt; 10000:     print("square:", i)     i = i ** 2</pre>                 | <b>Lambda</b> alternative syntax for one-liners<br><pre>cubed = lambda i: i ** 3 print("3 is ^", cubed(3))</pre>   |
| <b>list methods</b><br><pre>a = ["a", "b", 3] a.append(4) # ["a", "b", 3, 4] a.reverse() # [4, 3, "b", "a"]</pre>  | <b>For loop</b> Repeat for each item in iterable<br><pre>names = ["John", "Paul", "Q"] for name in names:     print("name:", name) for i in range(0, 100):     print("a:", a)</pre>         | <b>Try / except</b> Handle or ignore errors.<br><pre>try: big_number + 1 / 0 except Exception as e:     print("It broke:", e)</pre>  |
| <b>dict methods</b><br><pre>a = {"a": 1, "b": 2} a.get("c", 3) # 3 as default a.update({"d": 4}) # add more a.keys() # iterable of keys a.values() # ... of values a.items() # ... of both</pre> | <b>List comprehension</b> Create a new list while looping<br><pre>names = ["John", "Paul", "Q"] long_names = [     a.lower() for a in names     if len(a) &gt; 3 ] # ["john", "paul"]</pre> | <b>With</b> Execute code in a context<br><pre>with open("file.txt") as f:     f.write("test")</pre>  |
| <b>Prompt user</b><br><pre>name = input("Name? ") print("Hi ", name)</pre>   | <b>Interruption</b> Exit loops prematurely with break, skip to next iteration with continue<br><pre>a = "Some text for c.txt" open("a.txt", "w").write(a)</pre>                             | <b>Unpacking assignment</b> Assign to two or more, good for loops.<br><pre>x, y = [35, 15] pairs = [(10, 5), (8, 100)] for left, right in pairs:     print(left + right)</pre>   |
| <b>Read from file and convert to str</b><br><pre>a = open("file.txt").read() print("data:", a.decode("utf-8"))</pre>   |   |  |
| <b>Write to file creating if none</b><br><pre>a = "Some text for c.txt" open("a.txt", "w").write(a)</pre>  |   |  |
| Python (3 and later)   | kickstartcoding.com   | A cheatsheet by Kickstart Coding   |

## JavaScript for Pythonistas

Already know Python, and want to learn JavaScript (ES6+)? Print up the following cheatsheet to use as a reference.

## PYTHON

## JAVASCRIPT

```
# Variables
full_name = "Jane Hacker"
pi = 3.14

# Lists ("arrays" in JS)
names = ["John", "Paul", "Q"]

# Dicts (similar to "Objects")
translation = {
    "ola": "Hello",
    "oi": "Hi",
}

# For loops
for name in names:
    print("name:", name)

# While loops
x = 0
while x < 3:
    print("x:", x)
    x += 1

# If-statements
if full_name == "Jane":
    print("Hi, Jane!")
elif full_name == "Alice":
    print("Hey Alice")
else:
    print("Don't know you")

# List comprehension
long_names = [
    name.upper() for name in names
    if len(name) > 3
]

# Functions
def greeter(name):
    print("Hi", name)
    greeter("Bob")

# Lambda function
det = lambda x, y: x*x + y*y

# Conjunctions
if age < 18 and drink == "beer":
    print("Too young kiddo")

if age > 18 or drink == "coca":
    print("Great choice")

# Class syntax
class User(BaseUser):
    def __init__(self, name):
        self.name = name
        self.logged_in = False

    def log_in(self):
        self.logged_in = True

user = User("jshacker")
```

```
// Variables
let fullName = "Jane Hacker";
const pi = 3.14;

// Arrays ("lists" in Py)
let names = ["John", "Paul", "Q"];

// Objects (similar to "Dicts")
let translation = {
    ola: "Hello",
    oi: "Hi",
};

// For loop
for (let name of names) {
    console.log("name:", name);
}

// While loops
let x = 0;
while (x < 3) {
    console.log("x:", x);
    x++;
}

// If-statements
if (fullName === "Jane") {
    console.log("Hi, Jane!");
} else if (fullName === "Alice") {
    console.log("Hey Alice");
} else {
    console.log("Don't know you");
}

// Array processing (map & filter)
let longNames = names
    .filter(a => a.length > 3)
    .map(a => a.toUpperCase());

// Functions
function greeter(name) {
    console.log("Hi", name);
}
greeter("Bob");

// Arrow function expression
const det = (x, y) => x*x + y*y;

// Conjunctions
if (age < 18 && drink === "beer") {
    console.log("Too young kiddo");
}

if (age > 18 || drink === "coca") {
    console.log("Great choice");
}

// Class syntax
class User extends BaseUser {
    constructor(name) {
        this.name = name;
        this.loggedIn = false;
    }
    logIn() {
        this.loggedIn = true;
    }
}

let user = new User("jshacker");
```

```
// Destructuring object
let {ola, oi} = translation;
// ...and the reverse
const age = 3;
let info = {fullName, age};

// Combine obj, arrays with spread
const addedTranslations = {
    ...translation,
    gato: "cat",
};
const extra = [...names, "Mary"];
```

## LEGACY SYNTAX

```
// Loop through properties
for (var i in arr) {}

// C-style: Loop through numbers
for (var i = 0; i < 100; i++) {}

var a = 2; // function-scoped
a = 3; // globally scoped
```

## ASYNC &amp; CALLBACKS

```
// Callbacks (common in node.js)
fs.readFile("file.txt",
    (err, data) => {
        if (err) throw err;
        console.log(data);
    });

// Promises (common in browser)
fetch("http://site.com/api.json")
    .then(response => response.json())
    .then(data => {
        console.log("Resp:", data);
    })
    .catch(err => {
        console.log("error", err);
    });
```

asynchronous instead of passing  
"blocking" for a slow operation,  
the asynchronous approach is to  
put-off starting the slow operation,  
then call a function ("callback")  
at a later time to signal it's done  
callback A function passed as an argu-  
ment to be called later when an  
event is triggered  
promise Another popular way to do  
callbacks, with a .then syntax

## VARIABLE DECLARATION

```
let Declare a variable (block-scoped)
const Like let, cannot be reassigned.
```

## Python for JS Developers

Already know JavaScript, and want to learn Python 3? Print up the following cheatsheet to use as a reference.

## JAVASCRIPT

```
// Variables
let fullName = "Jane Hacker";
const pi = 3.14;

// Arrays ("lists" in Py)
let names = ["John", "Paul", "Q"];

// Objects (similar to "dicts")
let translation = {
  en: "Hello",
  es: "Hi",
};

// For loop
for (let name in names) {
  console.log("name:", name);
}

// While loops
let x = 0;
while (x < 3) {
  console.log("x:", x);
  x++;
}

// If-statements
if (fullName === "Jane") {
  console.log("Hi, Jane!");
} else if (fullName === "Alice") {
  console.log("Hey Alice!");
} else {
  console.log("Don't know you");
}

// Array processing (map & filter)
let longNames = names
  .filter(n => n.length > 3)
  .map(n => n.toUpperCase());

// Functions
function greeter(name) {
  console.log("Hi", name);
}
greeter("Bob");

// Arrow function expression
const dat = (x, y) => x*x + y*y;

// Conjunctions
if (age < 18 && drink === "beer") {
  console.log("Too young kiddo");
}

if (age > 18 || drink === "mads") {
  console.log("Great choice");
}

// Class syntax
class User extends BaseUser {
  constructor(name) {
    this.name = name;
    this.loggedIn = false;
  }
  login() {
    this.loggedIn = true;
  }
}

let user = new User("jshacker");
```

Python (3 and later)

## PYTHON

```
# Variables
full_name = "Jane Hacker"
pi = 3.14

# Lists ("Arrays" in JS)
names = ["John", "Paul", "Q"]

# Dicts (similar to "Objects")
translation = {
  "en": "Hello",
  "es": "Hi",
}

# For loops
for name in names:
  print("name:", name)

# While loops
x = 0
while x < 3:
  print("x:", x)
  x += 1

# If-statements
if full_name == "Jane":
  print("Hi, Jane!")
elif full_name == "Alice":
  print("Hey Alice!")
else:
  print("Don't know you")

# List comprehension
long_names = [
  name.upper() for name in names
  if len(name) > 3
]

# Functions
def greeter(name):
  print("Hi", name)
greeter("Bob")

# Lambda function
dat = lambda x, y: x*x + y*y

# Conjunctions
if age < 18 and drink == "beer":
  print("Too young kiddo")

if age > 18 or drink == "mads":
  print("Great choice")

# Class syntax
class User(BaseUser):
  def __init__(self, name):
    self.name = name
    self.loggedIn = False
  def login(self):
    self.loggedIn = True

user = User("jshacker")
```

kickstartcoding.com

## PYTHON FOR JS DEVELOPERS

### MORE PYTHON TYPES

sets "keys-only dicts", with operations

```
a = {"a", 1, 4, "b"}
b = {"a", "b"}
print(a - b) # {1, 4}
```

tuples: immutable list (array)

```
tpl = ("a", "b", "c")
# can't do: tpl["a"] = 2
```

### PYTHON FEATURES

template strings: Python 3.6+ only

```
phrase = f"name is {full_name}"
```

Ternary operator

```
b = 3 if len(a) > 3 else 0
```

in operator can be used with any collection (dict, string, list, etc)

```
if "Paul" in names:
  print("Found Paul")
```

Try / except: Code should throw exceptions when necessary, and use try/except to handle errors

```
try:
  third_name = names[3]
except Exception as e:
  pass # Handle ALL exceptions
except IndexError as e:
  pass # Only one error type
```

### GENERAL DIFFERENCES

Blocking Callbacks aren't used in Python. The blocking approach is to pause execution and use return values. Although not commonly in use, Python 3.5+ supports *async* via *async* and *await* syntax.

Object oriented programming: Python frameworks tend to use classic OOP more often than JS. Operator overloading: Python allows custom classes to use operators. Example: define a method called `__add__` to allow the `+` operator.

### ZEN OF PYTHON

*Simple is better than complex*  
*There should only be one way to do it*  
*Premature optimization is root of all evil*

A cheatsheet by Kickstart Coding

## React, Redux, and React Router

Learning React, Redux, and React Router can be tricky. This crams in important syntax and examples for common React and Redux coding patterns. Includes a Redux diagram, JSX syntax snippets, map and ternary operator examples, conditional rendering, action creator and reducer examples, React form events, and component syntax examples.



## REACT EXAMPLE CODE

```
import useState, useEffect from "react";
import { /App.css";
import ReactDOM from
"/images/envelope.png";
import Button from
"/components/Button";

function App() {
  // Define starting state
  const [message, setMessage] =
    useState("");
  const [chatlog, setChatlog] =
    useState([]);

  // Form elements need state
  // modifying functions
  function onChangeChange (ev) {
    setMessage(ev.target.value);
  }

  function sendMsg (ev) {
    setChatlog(
      [...chatlog, ...[message, ...]]);
  }

  // when you need something to
  // happen after first render
  useEffect(() => {
    alert("First render!");
  });

  // when you need something to
  // happen after every render
  useEffect(() => {
    alert("Just rendered!");
  }, []);

  // when you need something to
  // happen after every time
  // the chatlog state changes
  useEffect(() => {
    alert("chatlog changed!");
  }, [chatlog]);

  const count = chatlog.length;

  return (

    <div className="app">
      <div>{count} messages</div>
      {chatlog.map(text => {
        return <p>{text}</p>
      })}
      <input
        value={message}
        onChange={onChange} />
      <button onClick={sendMsg}>
        Send message</button>
      </div>

    </div>
  );
}
```

### DEFINING COMPONENTS

```
function Button(props) {
  return (
    <button className="Button"
      onClick={props.onClick}>
      {props.children}
    </button>
  )
}
```

## USEFUL REACT PATTERNS

### Conditional rendering

```
if (!props.test) {
  return "Empty...";
}

return (
  // full render here ...
);
```

### Using map to loop through data:

```
<div>{
  data_map((item, i) => {
    <p onClick={clicked}>
      {i}: {item}
    </p>
  })
}</div>
```

### Using ternary operators

```
<div>
  props.image ? (
    <img src={props.image} />
  ) : (
    <em>No image provided.</em>
  )
</div>
```

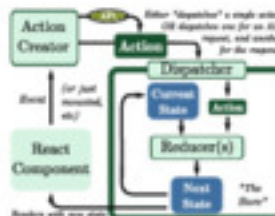
Using *ref* to incorporate legacy JS:

```
// At the top of the component
const [el, setEl] = useState('')

// In a useEffect call
useEffect(() => {
  $el().somePlugin();
}, [el]);

// Somewhere in JSX
<div ref={setEl} onClick={setEl}>
```

REACT, ROUTER, REDUX  
REACT REDUX



Action Creators (found in actions/)

```
const decrement = () =>
  ({type: INCREMENT});
const addToDo = (item) =>
  ({type: ADD_TODO, text: item});
```

Dispatching (load in components)

```
let action =
  addTodo(text)
dispatch(action);
```

Reducers (found in reducers/

```
const initialState = {
  count: 0,
  todoList: [],
};

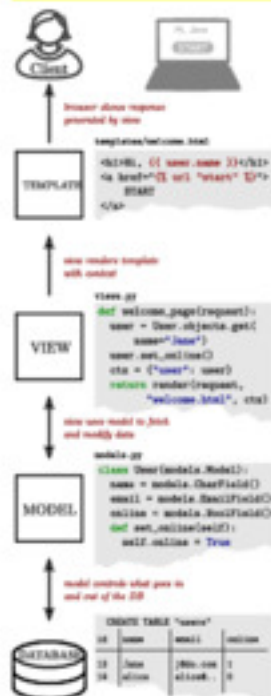
const todo = (state, action) => {
  switch (action.type) {
    case INCREMENT:
      return [...state, ...{
        count: state.count + 1,
      }];
    case ADD_TODO:
      return [...state, ...{
        todoList: [...todoList,
          action.text],
      }];
  }
}; // etc ...
```

## REACT ROUTER

```
<nav>
  <link to="/about/">About</link>
  <link to="/post/*postid+*/">
    Read More...</link>
</nav>
<main>
  <Switch>
    <Route path="/about/"
      component={About} />
    <Route path="/post/:id/"
      component={BlogPost} />
  </Switch>
</main>
```

## Django

Learning Django? Here are some common patterns of Django, along with a diagram showing how the Models-View-Template system fits together.



#### PYTHON MANAGE.PY WORK-FLOW

```
startapp # Scaffold django "app"
runserver # Run test server
shell # Enter Python shell
dbshell # Enter Database shell
showmigrations # Migration status
makemigrations # Gen model changes
migrate # Apply migrations to DB
```

#### MODELS.PY: DB SCHEMA

```
class Author(models.Model):
    # A single text field
    name = models.CharField(
        max_length=100)
    # In/Out ... on your models
    def __str__(self): # and return
        return self.name # its title

class Book(models.Model):
    # Use a "ForeignKey" for a
    # ManyToOne relationship
    author = models.ForeignKey(
        Author, on_delete="CASCADE")
    # Store when created and modified
    created = models.DateTimeField(
        auto_now_add=True)
    updated = models.DateTimeField(
        auto_now=True)
    # TextField is for long text.
    # null=True, blank=True allows
    # values of "" and None
    blurb = models.TextField(
        null=True, blank=True)
    # Multiple-choice fields in patterns:
    # "internal code, external label"
    CATEGORIES = [
        ("fiction", "Fiction"),
        ("nonfiction", "Non-fiction")]
    category = models.CharField(
        max_length=10,
        default="fiction",
        choices=CATEGORIES,
    )
    # Validators add custom checks
    num_stars = models.IntegerField(
        validators=[MaxValueValidator(5),
                    MinValueValidator(1)])
```

#### ORM: CRUD EXAMPLES

```
### CREATE: Save new book to DB
book = Book.objects.create(
    title="Oliver Twist",
    num_stars=5)
### READ: Get all fiction books
fiction_books = Book.objects(
    .filter(category="fiction")
    # Get all 4+ star books, newest first
    .filter(num_stars__gt=3)
    .order_by("-date")
)
### UPDATE: Change existing book(s)
book = Book.objects.get(title="1984")
book.num_stars = 5 # Update a property
book.save() # Save change to the DB
nonfiction = Book.objects.filter(category="nonfiction")
nonfiction.update(num_stars=0) # Update a
### DELETE: Delete one or more book(s)
book = Book.objects.get(title="1984")
book.delete() # Delete a single book
Book.objects.filter( # Delete multiple
    num_stars__lt=3).delete()
```

#### TEMPLATES

variables Use "." to access dict keys and properties and methods

```
<K2>{{ {{ user.username }}|</K2>
If
{% if age > 17 %}
    <p>You may continue.</p>
{% else %}
    <p>Too young.</p>
{% endif %}
```

```
for
{% for post in blog.posts %}
    <div>{{ post.title }}</div>
{% empty %}
    <no posts found</no>
{% endfor %}
```

```
extends & blocks Allow template
variations: replace "block" place-
holder in a base.html
{% extends "base.html" %}
{% block main_content %}
    <p>User: {{ user.name }}</p>
{% endblock main_content %}
```

```
Filters <p>{{ {{ name|upper }}</p>
```

```
include {% include "snippet.html" %}
using forms
```

```
<form action="" method="POST">
    {% csrf_token %} {{ form }}
    <button>Save</button> </form>
```

#### URLS.PY: ROUTING

```
from django.urls import path, include
from myapp import views
import accounts
urlpatterns = [
    path("/", views.welcome_page),
    path("/u/<int:uid>/", views.upage),
    include("accounts", namespace="accounts.urls"),
]
```

#### FORMS.PY: FORM VALIDATION

```
from django import forms
class NewUserForm(forms.Form):
    name = forms.EmailField()
```

#### VIEWS.PY: BUSINESS LOGIC

```
def welcome_page(request):
    return render(request,
        "hi.html", {})
def upage(request, uid):
    usr = Users.objects.get(id=uid)
    return render(request, "u.html", {
        "user": usr })
```

## react-hooks-useeffect

Tackling the new React Hook of useEffect? This React hook can do a lot more than just replace life-cycle methods. Here are a few common patterns and tricky gotchas:



## TERMINOLOGY

**useEffect** A function that adds the ability to perform side-effects from a function component.

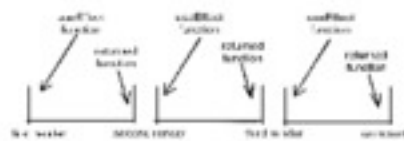
**mount** when a component is first added to a page

**unmount** when a component is removed from the page

## USEEFFECT ARGUMENTS

**first argument** The first argument to `useEffect` should be a function. Whatever happens in that function will get run when the component mounts and after each render.

**the return value of the first argument** The first argument to `useEffect` can also, optionally, return a "cleanup" function. If it does, the returned function will get run at the end of the render cycle, just before the next render (or unmount) happens.



useEffect lifecycle

**the second argument (optional)** By default, `useEffect`'s functions get run every time the component renders. If a second argument is provided, it should be an array. When an array is passed, `useEffect`'s functions get run only on the renders when at least one of the array's values have changed.

**passing an empty array** Because `useEffect`'s functions get run only on the renders when at least one of the array's values have changed, passing an empty array, `[]`, as the second argument is a way of saying "only run these functions on the first render and the unmount."

## USEEFFECT EXAMPLES

```
// Show the alert after every render
useEffect(() => {
  alert("This component just rendered!")
})
```

```
// Show the alert after the first render only
useEffect(() => {
  alert("This component just mounted!")
}, [])
```

```
// After first render AND after 'someValue' changes
useEffect(() => {
  alert("someValue just changed!")
}, [someValue])
```

## USEEFFECT EXAMPLES (WITH RETURN)

```
// Let's say we have a chat app with chatrooms.
// This code sets up a listener to display chat
// messages on receive. With this code, we will
// setup and then remove the listener on *every* render
useEffect(() => {
  listenForMessagesFrom(chatroomId, addToMessageList);

  return () => {
    stopListeningForMessagesFrom(chatroomId);
  };
})
```

```
// Same as above, except we're passing an empty array as
// the second argument to useEffect, so we only add the
// listener on the very first render, and only remove it
// right before unmounting. Much more efficient!
```

```
useEffect(() => {
  listenForMessagesFrom(chatroomId, addToMessageList);

  return () => {
    stopListeningForMessagesFrom(chatroomId);
  };
}, []); // we added the empty array on this line
```

```
// What if we change chatrooms, though? We don't want
// to display messages from our old chatroom after
// changing to a new one. So let's set it up to ALSO
// rerun the listener cleanup and setup code
// whenever chatroomId changes**
```

```
useEffect(() => {
  listenForMessagesFrom(chatroomId, addToMessageList);

  return () => {
    stopListeningForMessagesFrom(chatroomId);
  };
}, [chatroomId]); // we added chatroomId on this line
```

## COMMON GOTCHA

```
// THE GOTCHA: If you're not careful, calling a setter
// function inside useEffect can cause an infinite loop
const [count, setCount] = useState(0);
```

```
useEffect(() => {
  const savedCount = localStorage.get("savedCount");
  // Calling setCount re-renders the component
  // which means useEffect will get called
  // again, which means setCount will get
  // called again, and so on INFINITE LOOP
  setCount(savedCount);
});
```

```
// THE FIX: use that second useEffect argument to
// prevent unnecessary useEffect calls
const [count, setCount] = useState(0);
```

```
useEffect(() => {
  const savedCount = localStorage.get("savedCount");
  setCount(savedCount);
  // This little array argument right here is the fix!
  // Now this code will only run after the first render
}, []);
```

## More

More cheatsheets, corresponding to weekly Kickstart Coding curriculum.

## Contributing

- **How it's built:** These cheatsheets are written originally in Markdown, converted into LaTeX using pandoc and a custom pandoc LaTeX template (very messy), and then generates the PDF. All

---

this is tied together using the Bash script `build.sh`.

- **Writing your own:** Want to write your own printable coding cheatsheets? It's easy, as long as you know Markdown! Look at the existing `.md` source files for ideas, then follow the `DEVELOPMENT.md` for instructions on how to re-build the PDFs yourself.
- **License:** These cheatsheets and the scripts involved with their build process are (C) Kickstart Coding and released under the GPL 3.0