Ajuntament de Barcelona

Cibernàrium

Django: El framework web pels exigents amb deadlines

Capacitació Tecnològica per a Professionals i Empreses

Formador
Isaac Gallego Pla

Barcelonactiva

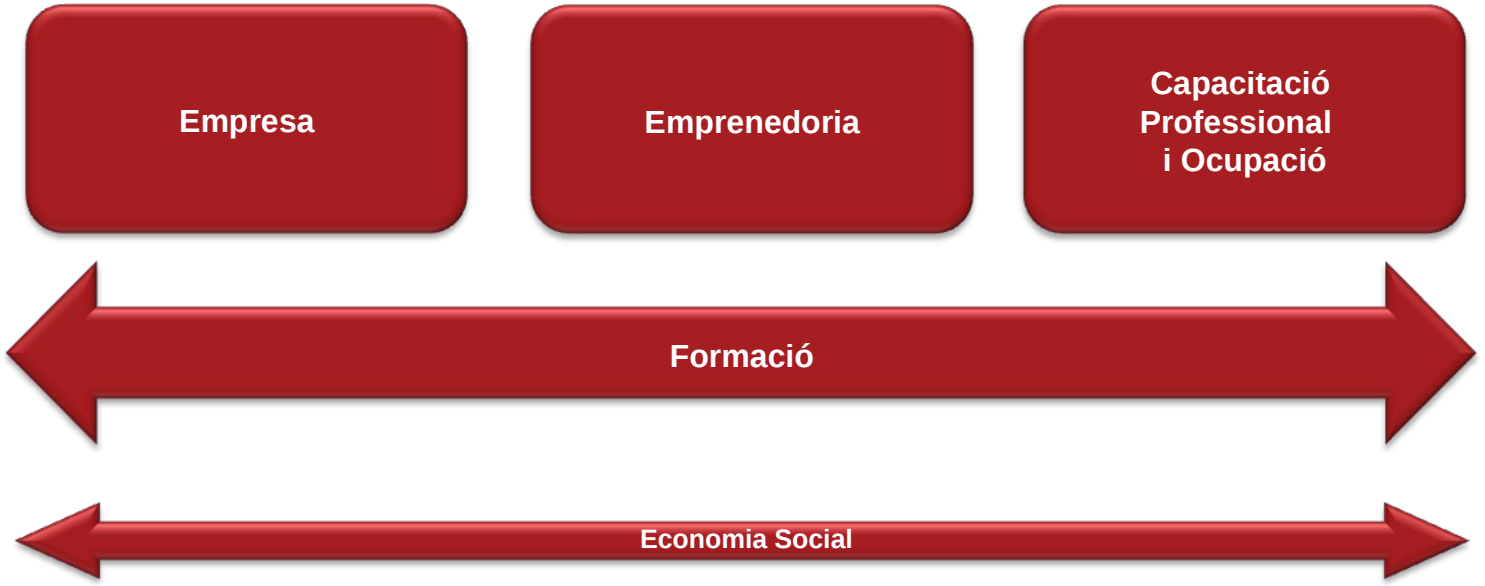
Barcelona Activa: Qui som?

Barcelona Activa, integrada en l'àrea d'Economia, Empresa i Ocupació, és l'organització executora de les polítiques de promoció econòmica de l'Ajuntament de Barcelona.

Des de fa 25

Barcelona Activa va ser guanyadora del Gran Premi del Jurat 2011, atorgat per la DG d'Empresa i Indústria de la Comissió Europea en el marc dels *European Enterprise Awards*, per la iniciativa empresarial més creativa i inspiradora d'Europa.

Àrees d'activitat de Barcelona Activa



Una xarxa d'Equipaments Especialitzats

Xarxa de Proximitat

13 antenes Cibernàrium a biblioteques
10 punts d'atenció en Ocupació

Temes de la sessió 1

- 1 – Introducció a Python
- 2 – Patrons de disseny
- 3 – Introducció a Django
- 4 – Projecte 1: El teu blog

1- Introducció a Python

- 1.1 – Què és Python?
- 1.2 – Breu història
- 1.3 – Variables
- 1.4 – Estructures de control
- 1.5 – Funcions i classes
- 1.6 – Útils
- 1.7 – Pràctica

1.1 – Què és Python

- Python és un llenguatge de programació interpretat, d'orientació generalista.
- No fa falta compilar (errors evaluats a l'inici i a runtime)
- Multiplataforma
- Gran comunitat que genera multitut de paquets third-party
- Ecosistema de paquets comparable al de Java

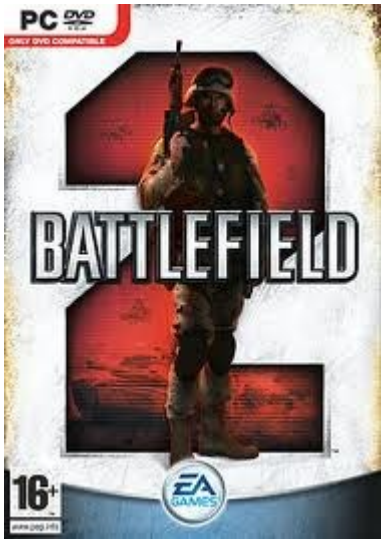
1.2 – Breu història

- Guido van Rossum
- Llenguatge de finals dels 80
- 1.0 – 1994
- Rep contribucions de molts llenguatges (Modula-3, Lisp, Haskell, Java...)
- Python 2.0 – Python Software Fundation Licence
- CPython, Jython, IronPython

  http://es.wikipedia.org/wiki/Python#Historia

Python – Qui l'utilitza?

Google



redhat

IBM

NASA

Exemple de codi

```
def get_available_changes(word1, word2, available_words, used = []):
    current_distance = check_word_distance(word1, word2)
    change_words = []
    for i in range(0, len(word1)):
        for j in letters:
            _aux = list(word1)
            _aux[i] = j
            change_words.append(''.join(_aux))
    change_words = set(change_words)
    change_words = filter(lambda a: a not in used, change_words)
    change_words = filter(lambda a: a in available_words, change_words)
    change_words = filter(lambda a: check_word_distance(a, word2) <= current_distance, change_words)
    change_words = sorted(change_words, key=lambda a: check_word_distance(a, word2))
    return change_words

def get_word_path(word1, word2, word_dict, words_used = []):
    if word1 == word2:
        return [words_used + [word2], 1]

    if not word1 in words_used:
        new_words_used = words_used + [word1]
    else:
        new_words_used = words_used
    available_words = get_available_changes(word1, word2, word_dict, new_words_used)
    if not available_words:
        return [words_used, -1]
    last_words_used = -1
    real_words_list = []
    code = -1
    for nword in available_words:
        last_words_list, code = get_word_path(nword, word2, word_dict, new_words_used)
        if code == -1:
            continue
        if len(last_words_list) == len(new_words_used)+code:
            real_words_list = last_words_list
            break
        if len(last_words_list) < last_words_used or last_words_used == -1:
            real_words_list = last_words_list
            last_words_used = len(real_words_list)
    if code != -1:
        return [real_words_list, code+1]
```

1.3 – Variables

- Existeixen les variables?
  - En python en diem Identificadors: no son variables propiament dites, sino referències a valors concrets

```
j = 3
s = j
```

- Sense tipat
- Multiassignació

```
a, b, c = func1()
a, b, c = d
```

1.3 – Variables

- Implementació CPython
  - Punters a regions de memòria
  - GIL: Global Interpret Lock
- Diccionaris

```
{'var1': 'val1', 'var2': 1}
```

- Llistes

```
[1,2,3,4,5]
```

- Sets

```
(1, "val1", 3.1415)
```

1.3 – Variables

- Strings
  - “val1”
- Unicode
  - u”val1”
- Diferències en l'interpretació. Normalment és preferible utilitzar sempre unicode

1.4 – Estructures de control

- Condicionals
  - If – elif – else
  - Acabades amb el símbol “:”

```
if a < b:
    process_a()
elif b > c:
    process_b()
else:
    process_c()
```

1.4 – Estructures de control

- Loops
  - While

```
while a > b:
    do_something()
```

  - For

```
for item in item_list:
    do_something(item)
```

1.4 – Estructures de control

- Modificadors
  - Continue: executa la següent iteració del loop
  - Break: talla l'execució del loop
  - Pass: paraula clau que designa la implementació nul·la

1.4 – Estructures de control

- Control d'excepcions
  - Try – except

```
try:
    code()
except Exception:
    code_exception()
except AnotherException:
    code_another_exception()
except:
    code_last_exception()
```

1.5 – Funcions i classes

- Funcions

```
def func_1(arg1, arg2, arg3 = 2):
```

- Crida de funcions

```
func_1(1, 2, 3)
func_1(1, 2)
func_1(arg2 = 1, arg3 = 2, arg1 = 1)
```

- Funcions anònimes

```
f1 = λ a, b: a+b
f1(2, 3)
```

- Retorn: return

1.5 – Funcions i classes

- Classes

```
class c1(object):
class c2(c1, c3):
```

- Instanciació de classes

```
i1 = c1()
i2 = c3(1, “test”, 3, var4=5)
```

1.5 – Funcions i classes

- Self
  - Punter a la instància de classe
  - Primer argument dins cada funció d'instància de l'objecte

```
class C1(object):

    def myfunc(self):
        do_something

    def myfunc2(self, arg1, arg2):
        do_something
```

1.5 – Funcions i classes

- Elements privats
  - No hi ha concepte de privat o públic
  - Convenció amb __nom
- Constructor __init__(self)

```
def __init__(self, arg1, arg2):
    constructor_code
```

- Altres elements privats
  - __name__, __module__, __str__, __dict__ ...

1.6 – Útils

- Indentació
  - Python funciona amb indentacions, no amb brackets

http://docs.python.org/2/library/

- built-in functions
http://docs.python.org/2/library/functions.html
- __init__.py

  Fitxer físic que indica que el directori és un package de python

1.6 – Útils

- Concepte de mòdul
  - Fitxer físic que conté codi de python
  - Importable
- Mòdul executable
  - Tot aquell que té la següent declaració

```
If __name__ == “__main__”:
    code
```

- Interpret interactiu
  - Executa en runtime codi de python
  - En consola, executem el següent

```
python
```

1.6 – Útils

- Print per l'stdout

  print “linia”

- Strings enriquits per interpolació

```
var1 = u“Això és una %s. Número %i”
print var1 % (u“funció”, 45)
```

- Definir codificació del mòdul
  - A l'inici del fitxer

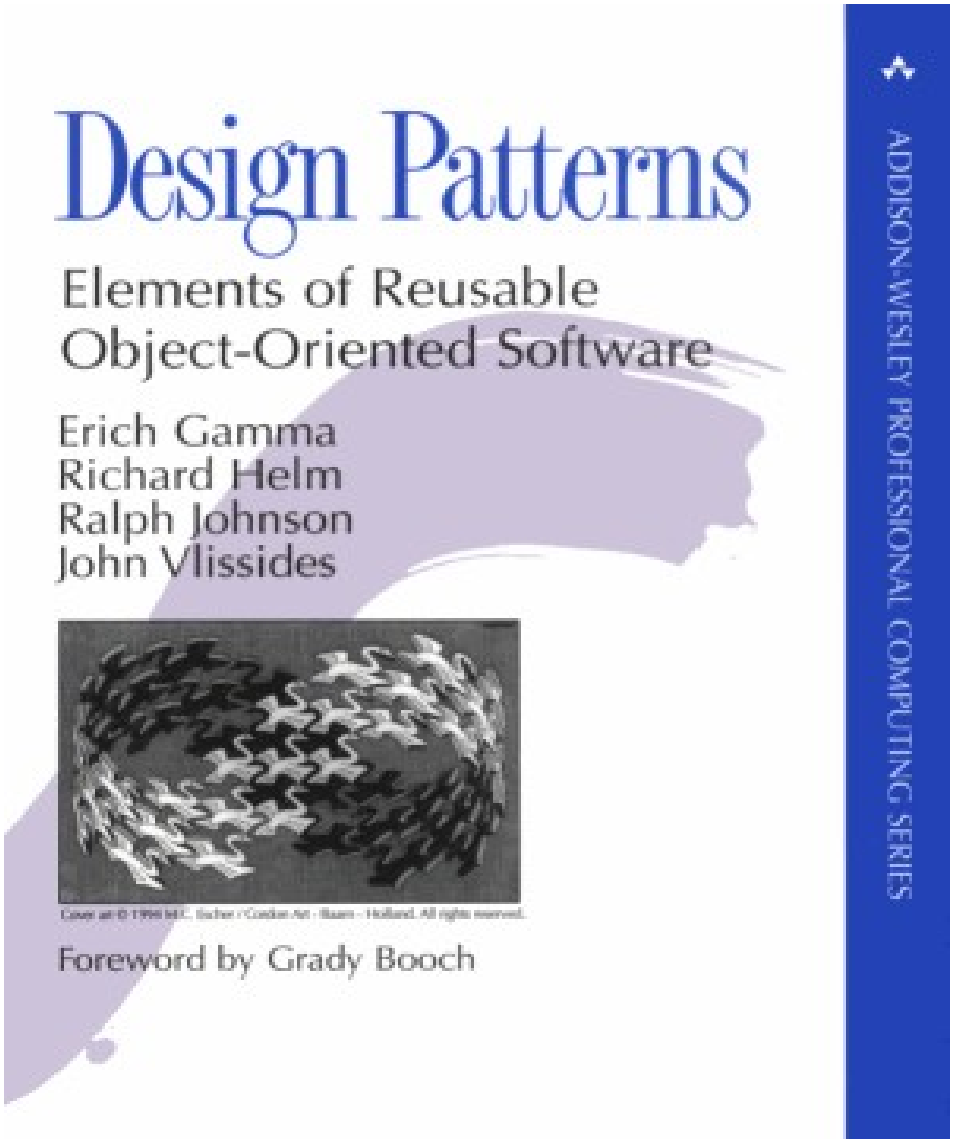
    # -*- coding: utf-8 -*-
  - Declarem així que la codificació és utf-8

1.7 – Pràctica

- Creació d'un mòdul python executable amb:
  - Una classe anomenada TestClass que conté
    - 3 variables anomenades p1, p2 i p3
      - P1 forçarem a que sigui float
      - P2 forçarem a que sigui enter
      - P3 forçarem que sigui string (o unicode)
    - 2 funcions anomenades adder i divider
      - Adder: retorna el sumatori de p1 i p2
      - Divider: retorna la divisió de p1 i p2
    - Funció constructor, on pasarem les 3 variables a l'inici
  - Un main on crear una instància de TestClass, i printar per pantalla un text amb l'execució de les dues funcions adder i divider

2 – Patrons de disseny

- 2.1 – Què son?
- 2.2 – MVC



2.1 – Què son?

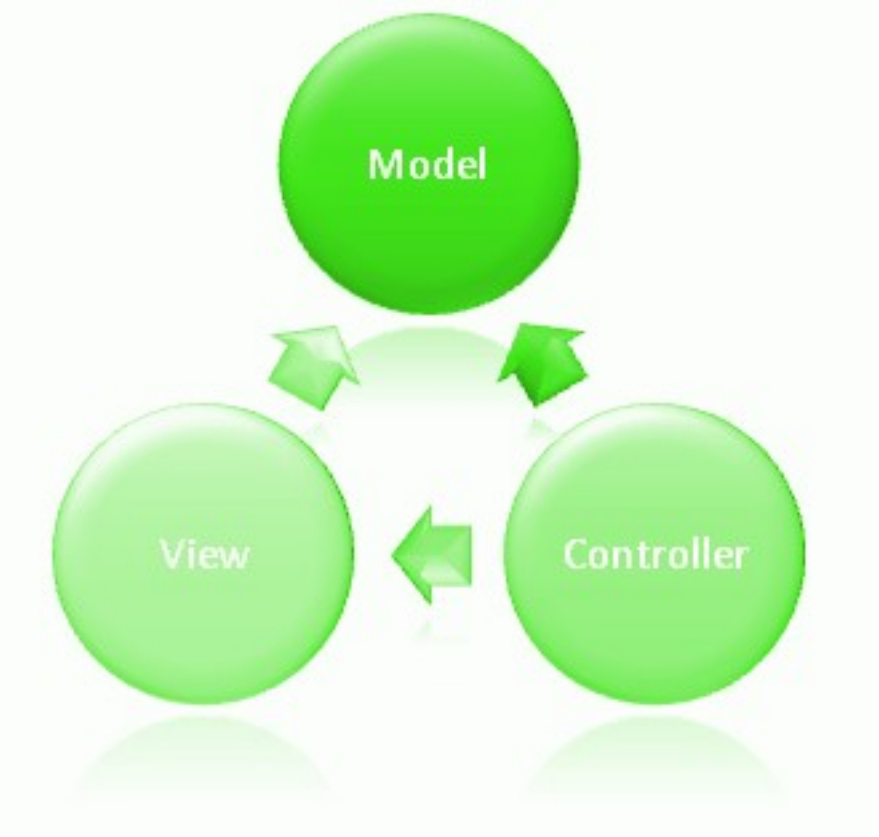
- Els patrons de disseny son, bàsicament, conjunts de bones pràctiques i solucions robustes a problemes coneguts
- Es poden prendre com a plantilles
- N'hi ha de diverses categoríes
  - De creació: sobre el procés de creació d'objectes
  - D'estructura: sobre la composició de classes/objectes
  - De comportament: sobre l'interacció i el repartiment de responsabilitats

2.1 – Què son?

- Exemples
  - Strategy: Encapsular algoritmes relacionats i fer-los intercanviables, segons una selecció basada en l'objecte en qüestió.
  - Observer: Definir una notificació automàtica d'una relació 1:n, de manera que el canvi d'estat de l'objecte target notifiqui automàticament a la resta i aquests quedin actualitzats.
  - Singleton: Assegurem que una classe té una sola instància i es proporciona un punt d'accés global a ella.

2.2 – MVC

- Model – Vista – Controlador
  - Separació entre la lògica de negoci, la capa de dades i la visualització d'aquestes
  - Bones pràctiques de reutilització
  - Utilitzat en molts dels actuals frameworks web per el seu paralelisme amb l'infraestructura client – ser
  - Interfície per capes



2.2 – MVC

- **Model**: capa de dades. La interacció amb les dades i el seu model d'emmagatzemament es troben en aquesta capa
- **Controlador**: capa de lògica de negoci. Aqui encapsulem tots els càlculs necessàris per donar resposta a la petició efectuada
- **Vista**: capa de visibilitat. En aquest punt mostrem, de la manera desitjada, les dades calculades.

3 – Introducció a Django

- 3.1 – Introducció
- 3.2 – Models
- 3.3 – Vistes
- 3.4 – URLs
- 3.5 – Templates

3.1 – Introducció

- Programació àgil: orientació a prototipatge
- Framework: gran nombre d'eines, amb un alt grau de complementarietat, facilitant la implementació d'aplicacions web
- Eines
  - Servidor de desenvolupament
  - Shell interactiva
  - Altres comandes de sistema

3.1 – Introducció

- Configuració
  - **djangoadmin.py**: script global de sistema per la crida de comandes
  - **manage.py**: script local de projecte per la crida de comandes
  - **settings.py**: fitxer de configuració del projecte.
  - **wsgi.py**: fitxer de configuració wsgi. Utilitat pels deploys.

Django – Qui l'utilitza?

DISQUS

NATIONAL GEOGRAPHIC™

NASA

The New York Times

Barcelonactiva

Pinterest

The Washington Post

3.2 – Models

- Es basa en un ORM pròpi
- Possibilitat de canviar d'ORM (com SQLAlchemy)
- Tot comença amb django.db.models.Model
- Es defineix una classe per cada model, i es divideix en
  - Camps
  - Relacions

3.2 – Models

- Camps
  - S'obtenen a partir de la mateixa declaració models
  - Cadascun es defineix com a atribut del model pare
  - Exemples:

```
v1 = models.CharField(max_length = 100)
v2 = models.DateTiemField()
```

https://docs.djangoproject.com/en/dev/ref/models/fields/

3.2 – Models

- Relacions
  - Definició igual als camps, amb semàntica diferent
  - Tipus de relacions
    - Clau forànea

```
models.ForeignKey("ModelName")
```

    - Un a un

```
models.OneToOneField("ModelName")
```

    - Molts a molts

```
models.ManyToManyField("ModelName")
```

  - Molt important l'atribut "related_name", ja que defineix el nom d'accés per el model complementari

3.2 – Models

- Funcions
  - Es defineixen totes les funcions que necessiti el model (de fet, no és més que un objecte de Python)
- Meta
  - Definició de metacaracterístiques de la classe

    class Meta:
  - Exemples:

    ordering

    unique_together

3.2 – Models

- Sincronització amb la BBDD
  - Validació dels models

    python manage.py validate
  - Generació del SQL

    python manage.py sql APP_NAME
  - Sincronitzar amb la BBDD

    python manage.py syncdb

3.2 – Models

- Cerques
  - Totes sobre el manager
    - Per defecte, objects
  - Retorna un queryset
  - Diferents funcions de cerca
    - filter
    - get
    - exclude
  - Totes poden rebre paràmetres de lookup

3.2 – Models

- Paràmetres de lookup
  - Son els paràmetres que abstreuen les cerques
  - La seva sintaxi va precedida de __
  - Alguns exemples:
    - in
    - lt
    - gt
    - contains

3.2 – Models

- Exemple de model

```
class MyModel(models.Model):
    attr1 = models.CharField(max_length = 100)
    attr2 = models.IntegerField(null = True)
    rel = models.ForeignKey("OtherModel",
                            related_name = "mymodels")
    class Meta:
        ordering = ["attr1", "attr2"]
```

3.2 – Models

- Exemples de cerca

```
MyModel.objects.all()
MyModel.objects.filter(attr1 = “test”)
MyModel.objects.filter(attr1__contains = “test”)
OtherModel.objects.filter(mymodels__attr2 = 3)
MyModel.objects.get(id = 3)
```

3.2 – Models

- Afegir noves instàncies

```
mm = MyModel(attr1 = "Test", attr2 = 1)
mm.save()
```

- Eliminar instàncies

```
mm.delete()
```

- Accedir als camps / relacions / funcions

```
mm.attr1
mm.rel.attr3
```

- Son objectes Python normals!

3.3 – Vistes

- En Django, una vista és l'equivalent al controlador en el MVC
- Son funcions Django simples (tot i que en les últimes versions, també hi han vistes basades en classes)
- La interacció és simple: rep un objecte HttpRequest i retorna un altre objecte HttpResponse
- Dins HttpRequest conté tota la informació relacionada amb la petició

3.3 – Vistes

- Entrada de dades
  - Com a paràmetre de la URI
  - Com a paràmetre del tipus de petició
    - Principalment: GET / POST
- Sortida de dades
  - django.http.HttpResponse
  - Útils
    - django.http.HttpResponseRedirect
    - django.shortcuts.render_to_response

3.3 – Vistes

- Formats de sortida
  - Indicat com a content_type del HttpResponse
  - Per defecte: HTML
  - Facilment canviem de format. Exemple:
    HttpResponse(json.dumps(response),
    content_type = “application/json”)

3.3 – Vistes

- Decoradors
  - Comuns a totes les funcions python
  - Son wrappers de funcionalitat
  - Molt utilitzats com a útils a les vistes

    @login_required

    @csrf_exempt

3.3 – Vistes

- Exemple

```
@login_required
def view1(request):
    param1 = request.GET.get("param1", "default")
    return render_to_response("template1.html",
                              RequestContext(request,
                                             {}
                                             )
                              )
```

3.4 – URLs

- Fitxer de configuració urls.py
- Pot estar dividit en diversos fitxers, i només s'han d'incloure en el principal
- Defineix les URLs accessibles del sistema
- Sistema flexible amb expressions regulars
- django.conf.urls.patterns

3.4 – URLs

- Exemple

```
from myapp import views
urlpatterns = patterns("",
    url("^url1/path/$", views.view1, name="view1"),
    url("^url2/", include("myapp.urls")),
    )
urlpatterns += other_urlpatterns
```

3.5 – Templates

- Plantilles: equivalent de vista en el MVC
- Mescla d'elements de vista (HTML normalment) amb un pseudollenguatge
- Pseudollenguatge de templates basat en tags
- 2 notacions:

  {{ var }}: *mostra per pantalla el contingut de var*

  {% tag1 %}: *executa tag1*

3.5 – Templates

- Tags principals

```
{% extends “template_name” %}
{% includes “template_name” %}
{% block block_name %}
-
{% endblock %}
{% if cond %}
-
{% elif cond %}
-
{% else %}
-
{% endif %}
{% for item in collection %}
-
{% endfor %}
```

3.5 – Templates

- Exemple:

```
{% extends “base.html” %}
{% block title %}
Title test
{% endblock %}
{% block content %}
<h1>Test</h1>
{% block content %}
```

4 – Projecte 1: El teu blog

- Projecte web 1: El teu blog
- Descarregueu el projecte de la següent URI

  https://mega.co.nz/#!Ic4mXbxK!ZLcDtjx6wUUP7sNbuWrqOHD5U00hulaYWMD5GKVQmYM
- Aplicació senzilla: es tracta d'un blog personal. En ell podrem veure tots els conceptes estudiats fins ara.
- El projecte ve preconfigurat per facilitar la vostra feina (Però compte! No perdeu l'oportunitat de revisar el fitxer settings.py)

4 – Projecte 1: El teu blog

- Models:
  - Post: ordenat per defecte desc. per data publicació
    - Títol
    - Entradeta
    - Cos
    - Data de creació – Ha d'assignar-se automaticament
    - Data de publicació
  - Enllaç: ordenat per defecte asc. per nom
    - Post – Relació amb post
    - URL
    - Nom

4 – Projecte 1: El teu blog

- Vistes
  - Llistat de posts: es mostra títol i entradeta de cada post
  - Interior de post: es mostra títol, data de publicació, cos i llistat d'enllaços. Al final, afegim un enllaç de tornada al llistat de posts.

  ** Imprescindible fer ús d'herència de templates, i per tant, reutilitzar l'esquema general*

4 – Projecte 1: El teu blog

- Ampliació (Opcional):
  - Afegir comptador de visites a Post
  - Crear un sidebar
  - Mostrar en el sidebar els 3 Posts més visitats

Django

Gràcies i fins la propera sessió!

Barcelonactiva

Ajuntament de Barcelona

bcn.cat/barcelonactiva
bcn.cat/cibernarium