Exercice 1: Régression linéaire

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

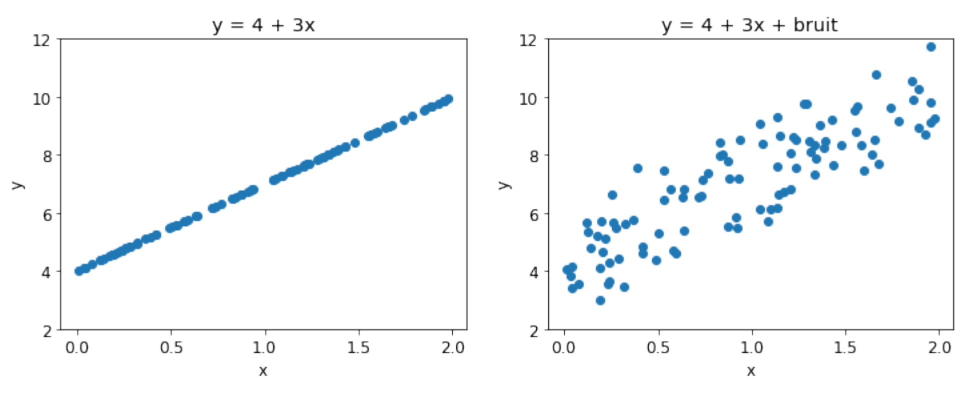
1 & 2- Génération d'un ensemble de données aléatoires

In [2]:
```
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y1= 4 + 3 * X
y = 4 + 3 * X + np.random.randn(100, 1)
```

3- Représentation graphique de la dispersion de y en fonction

In [3]:
```
plt.rcParams.update({'font.size': 12})
fig = plt.figure(figsize=(12, 4))
plt.subplot(121)
plt.scatter(X,y1)
plt.xlabel("x")
plt.ylabel("y")
plt.ylim(2,12)
plt.title("y = 4 + 3x")
plt.subplot(122)
plt.rcParams.update({'font.size': 12})
plt.scatter(X,y)
plt.xlabel("x")
plt.ylabel("y")
plt.title("y = 4 + 3x + bruit")
plt.ylim(2,12)
```

Out[3]:
```
(2.0, 12.0)
```



4- Implémentation analytique

In [4]:
```
# On doit ajouter x0 = 1 à chaque observation
# Ensuite on implémente l'équation normale

X_b = np.c_[np.ones((100, 1)), X]
theta_hat = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
theta_hat
```

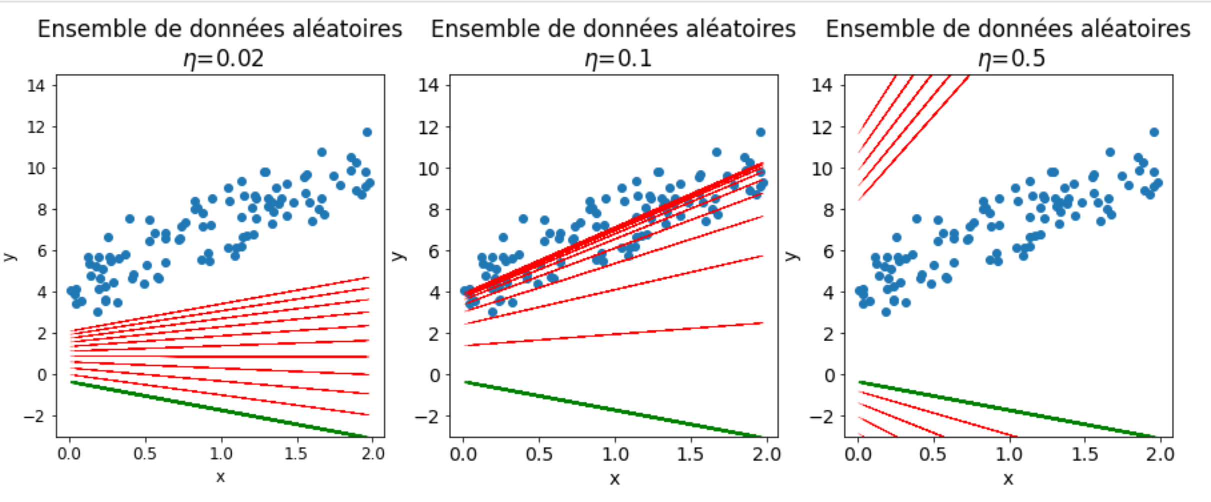
Out[4]:
```
array([[4.22215108],
       [2.96846751]])
```

On obtient les valeurs de $\theta_0 = 4.10$ et $\theta_1 = 2.77$. L'équation de régression linéaire est alors y_pred = theta_hat[0] + theta_hat[1] * X c-à-d y= 4.22 + 2.96 x

6 & 7 - Impact du pas d'apprentissage

In [5]:
```
# Fonction utilisé pour la représentation de la droite de régression
def DrawTheta(X,Y,eta,Theta,X_b,m,Color,LineWidth):
     # Plot init 'Theta'
    gradients = 2/m * X_b.T.dot(X_b.dot(Theta) - Y)
    Theta = Theta - eta * gradients
    y_pred = Theta[0] + Theta[1] * X
    plt.plot(X,y_pred, Color, linewidth=LineWidth)
    return Theta
```

In [6]:
```
# On doit ajouter x0 = 1 à chaque observation
# Fixer la valeur de l'hyperparamètre qui correspond au nombre d'itération.
# Le pas d'apprentissage étant un hyperpamètre mais qui sera variable
# thetaInit est une initialisation aléatoire des valeurs de theta
# La représentation graphique est réalisée pour les 10 première itérations

m=100
n_iterations = 1000
X_b = np.c_[np.ones((100, 1)), X]
List_eta=[0.02, 0.1, 0.5]
thetaInit = np.random.randn(2,1)

fig = plt.figure(figsize=(15, 5))

for i, eta in zip(np.arange(len(List_eta))+1, List_eta):
    theta=thetaInit

    plt.subplot(1, 3, i)
    plt.rcParams.update({'font.size': 14})
    plt.scatter(X,y)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("Ensemble de données aléatoires\n " + r'$\eta$='+str(eta))

    # Plot init 'Theta'
    y_pred = thetaInit[0] + thetaInit[1] * X
    plt.plot(X,y_pred, 'g--', linewidth=2)

    for iteration in range(n_iterations):
        if (iteration < 10):
            theta=DrawTheta(X,y,eta,theta,X_b,m,'r',0.5)

    plt.ylim(-3, 14.5)
    plt.plot()
```



8-Modèle de régression avec la classe SGDRegressor

In [7]:
```
# Modèle linéaire ajusté en se basant sur un descente
# de gradient stockastique
from sklearn.linear_model import SGDRegressor

sgd_reg = SGDRegressor(max_iter=1000, tol=1e-3, penalty=None, eta0=0.1)
sgd_reg.fit(X, y.ravel())
sgd_reg.intercept_, sgd_reg.coef_
```

Out[7]:
```
(array([4.17429195]), array([2.90225601]))
```

On obtient les valeurs de $\theta_0 = 4.10$ et $\theta_1 = 2.77$. L'équation de régression linéaire est alors y_pred = theta_hat[0] + theta_hat[1] * X c-à-d y= 4.17 + 2.90 x

Excercice2 : Regression polynomiale

In [8]:
```
import numpy as np
import numpy.random as rnd
import numpy as np
import matplotlib.pyplot as plt
```

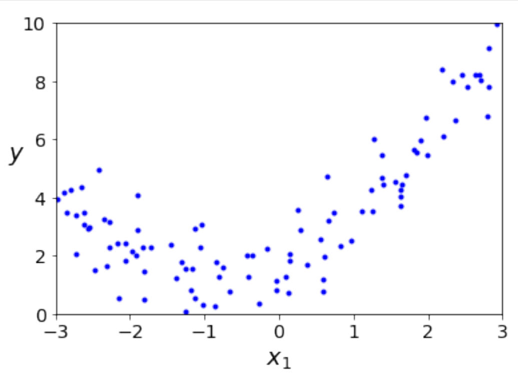
In [9]:
```
np.random.seed(42)
```

1 & 2 - Génération des données

In [10]:
```
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X**2 + X + 2 + np.random.randn(m, 1)
```

3- Représentation de y en fonction de x

In [11]:
```
plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.axis([-3, 3, 0, 10])
plt.show()
```



4- Génération d'un modèle de caractéristiques polynomiaux

In [12]:
```
# Génère une nouvelle matrice de caractéristiques composée
# de toutes les combinaisons polynomiales des caractéristiques
# avec un degré inférieur ou égal au degré spécifié.
# Dans notre cas degree=2

from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X)
```

In [13]:
```
# Vérification avec une valeur scalaire de X
X[0]
```

Out[13]:
```
array([-0.75275929])
```

In [14]:
```
# On obtient un vecteur à deux valeurs qui correspondent
# à X[0] et X[0]^2
X_poly[0]
```

Out[14]:
```
array([-0.75275929,  0.56664654])
```

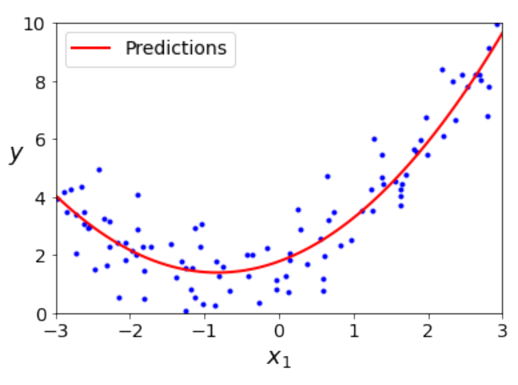
5- Modèle de régression polynomial

In [15]:
```
# On consiste à ajouter les puissances de chacune des
# variables comme nouvelles variables, puis d'entraîner
# un modèle linéaire sur ce nouvel ensemble de données

from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
lin_reg.intercept_, lin_reg.coef_
```

Out[15]:
```
(array([1.78134581]), array([[0.93366893, 0.56456263]]))
```

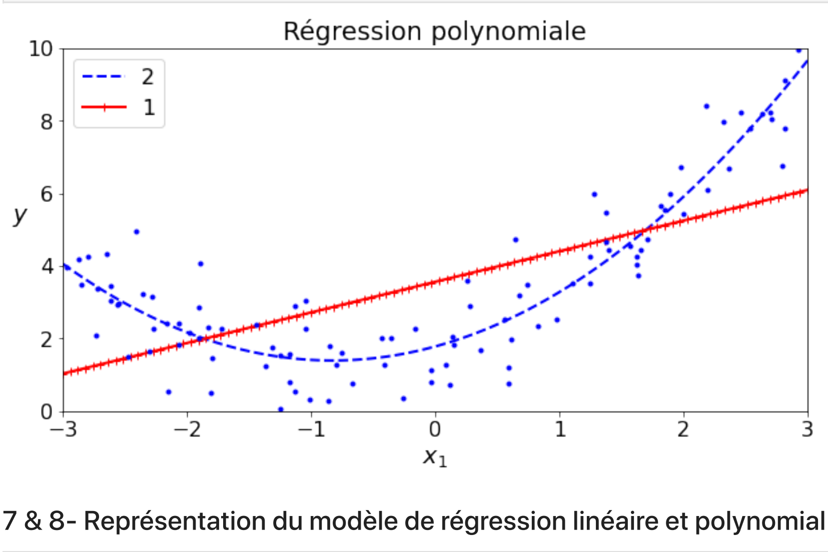
On obtient les valeurs de θ_0, θ_1 et θ_2. L'équation de régression polynomiale est alors y_pred = theta_hat[0] + theta_hat[1] *X + theta_hat[2]* X^2

In [16]:
```
X_new=np.linspace(-3, 3, 100).reshape(100, 1)
X_new_poly = poly_features.transform(X_new)
y_new = lin_reg.predict(X_new_poly)
plt.plot(X, y, "b.")
plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14)
plt.axis([-3, 3, 0, 10])
plt.show()
```



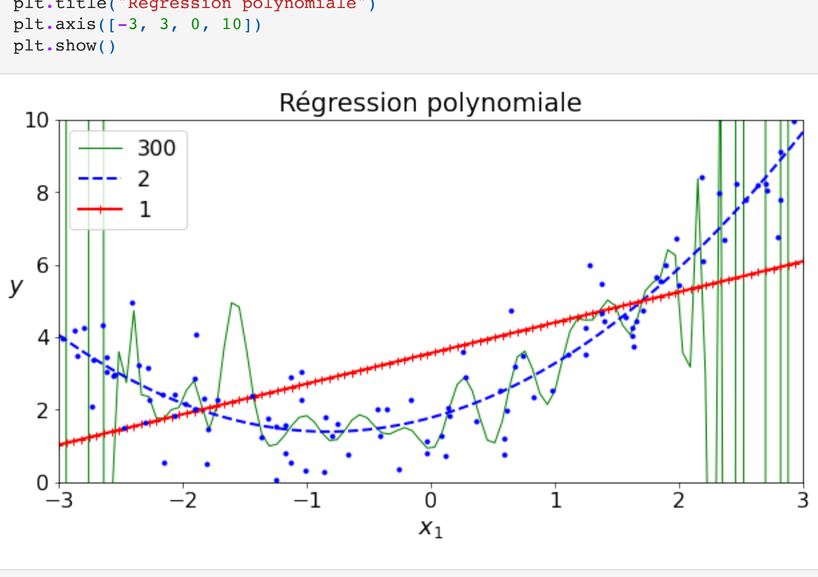
6- Représentation du modèle de régression linéaire et polynomial

In [17]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

dict=["a","b"]

plt.rcParams.update({'font.size': 16})
fig = plt.figure(figsize=(10, 5))
for style, width, degree in (("b--", 2, 2), ("r-+", 2, 1)):
    polybig_features = PolynomialFeatures(degree=degree, include_bias=False)
    std_scaler = StandardScaler()
    lin_reg = LinearRegression()
    polynomial_regression = Pipeline([
            ("poly_features", polybig_features),
            ("std_scaler", std_scaler),
            ("lin_reg", lin_reg),
        ])
    polynomial_regression.fit(X, y)
    y_newbig = polynomial_regression.predict(X_new)
    plt.plot(X_new, y_newbig, style, label=str(degree), linewidth=width)


plt.plot(X, y, "b.", linewidth=3)
plt.legend(loc="upper left")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.title("Régression polynomiale")
plt.axis([-3, 3, 0, 10])
plt.show()
```



7 & 8- Représentation du modèle de régression linéaire et polynomial d'ordre 2 et d'ordre 300

In [18]:
```
from sklearn.pipeline import Pipeline

dict=["a","b"]

plt.rcParams.update({'font.size': 16})
fig = plt.figure(figsize=(10, 5))
for style, width, degree in (("g-", 1, 300), ("b--", 2, 2), ("r-+", 2, 1)):
    polybig_features = PolynomialFeatures(degree=degree, include_bias=False)
    std_scaler = StandardScaler()
    lin_reg = LinearRegression()
    polynomial_regression = Pipeline([
            ("poly_features", polybig_features),
            ("std_scaler", std_scaler),
            ("lin_reg", lin_reg),
        ])
    polynomial_regression.fit(X, y)
    y_newbig = polynomial_regression.predict(X_new)
    plt.plot(X_new, y_newbig, style, label=str(degree), linewidth=width)


plt.plot(X, y, "b.", linewidth=3)
plt.legend(loc="upper left")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.title("Régression polynomiale")
plt.axis([-3, 3, 0, 10])
plt.show()
```



In [ ]:

In [ ]: