Project 1

A naive approach to population dynamics is to say that the rate of growth of a population is proportional to the population. Intuitivley, this makes sense because we would expect that the larger the population, the faster the rate of growth rate. However, there is an issue with this model as we will see below.

In the cell below, determine the differential equations that models a population whose growth rate satisfies this model. Let $P(t)$ denote the population at time t. So your equation will be $\frac{dP}{dt} = XXX$ and you have to determine what XXX is. Give a short statement that "derives" your differential equation (this is going to be *very* short).

Your answer here: At any time t the rate of change of the population is proportional to the population. dp/dt = kp

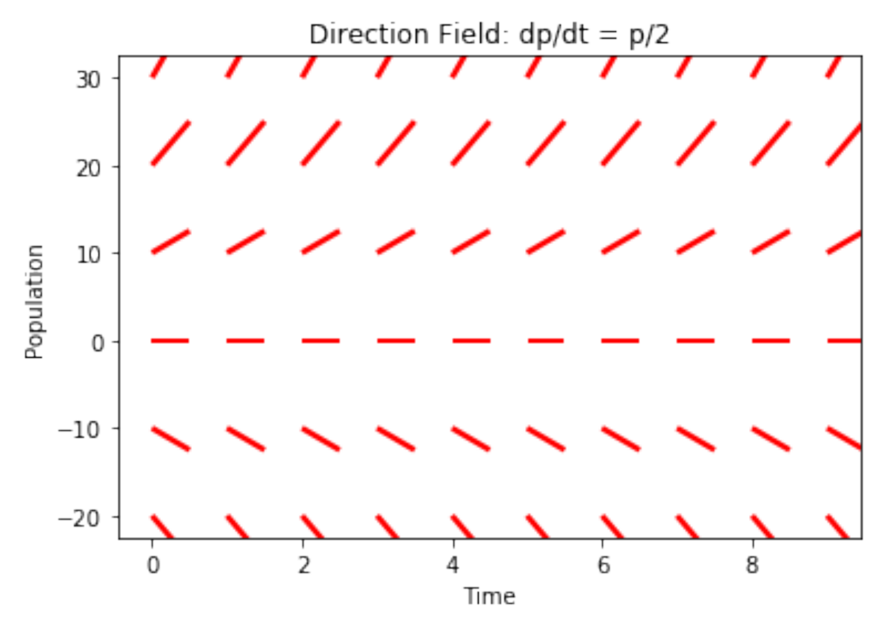
For constants of proportionality $.5$ and 1, in the two cells below, draw a direction field for the ODE corresponding to these specific values.

In [27]:
```
# Import statements
import numpy as np
import matplotlib.pyplot as plt

# Define the function
def f(t,x):
    return x/2

# (b-a)/h gives you the length of the array
# t = time; x = population
t = np.arange(0, 10, 1)  # Array from 0->9, with step size of 1
x = np.arange(-20, 40, 10)  # Array from -20->39, with step size of 10
#x = np.append(x, 0)  # equilibrium for the ODE
#x.sort() #line sorts the x array
T, X = np.meshgrid(t, x)

dXdT = f(T, X)
#U = (1 / (1 + dXdT**2)**.5)*np.ones(T.shape)
#V = (1 / (1 + dXdT**2)**.5)*dXdT
U = np.ones(T.shape) # Normalizes the arrows to see about where equilibriu
m is
V = dXdT
plt.figure() # Creates a figure on which to draw
plt.title('Direction Field: dp/dt = p/2')  # Gives vector a title
plt.xlabel('Time')  # X-axis label
plt.ylabel('Population')  # Y-axis label
Q = plt.quiver(T, X, U, V,  angles='xy', headlength=0, headwidth=1, scale_
units='xy', scale=2, color='red')
```

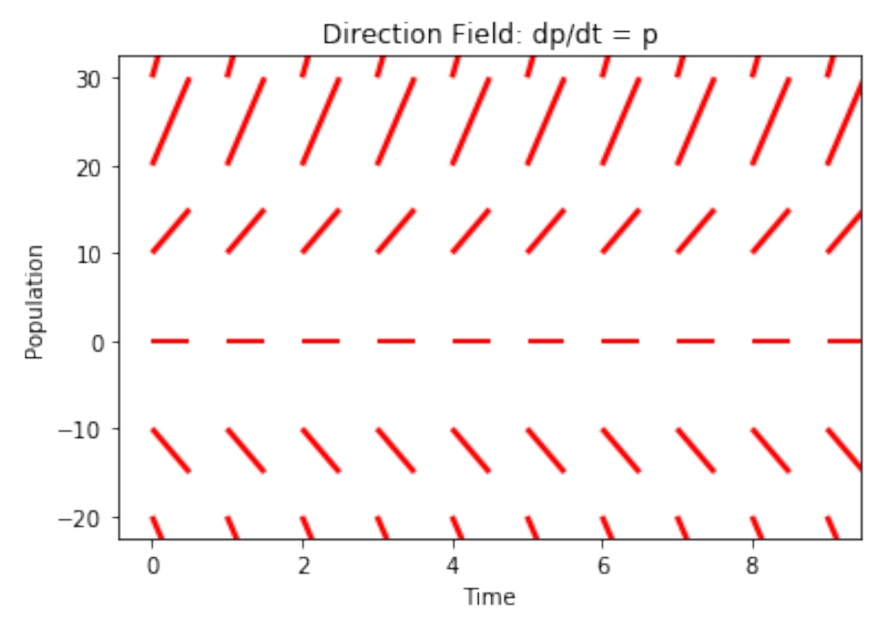


In [28]:
```
# Import statements
import numpy as np
import matplotlib.pyplot as plt

# Define the function
def f(t,x):
    return x

# (b-a)/h gives you the length of the array
# t = time; x = population
t = np.arange(0, 10, 1)  # Array from 0-9, with step size of 1
x = np.arange(-20, 40, 10)   # Array from 0-0, with step size of 0
#x = np.append(x, -49)  # equilibrium for the ODE
#x.sort() #line sorts the x array
T, X = np.meshgrid(t, x)

dXdT = f(T, X)
#U = (1 / (1 + dXdT**2)**.5)*np.ones(T.shape)
#V = (1 / (1 + dXdT**2)**.5)*dXdT
U = np.ones(T.shape) # Normalizes the arrows to see about where equilibriu
m is
V = dXdT
plt.figure() # Creates a figure on which to draw
plt.title('Direction Field: dp/dt = p')  # Gives vector a title
plt.xlabel('Time')  # X-axis label
plt.ylabel('Population')  # Y-axis label
Q = plt.quiver(T, X, U, V,  angles='xy', headlength=0, headwidth=1, scale_
units='xy', scale=2, color='red')
```



Using the direction fields from above explain below why there is a problem with this model. (Be sure that you are using information from the direction fields above.)

Your answer here: The model above is incorrect because it displays a population that does not decrease due to extrinsic factors, also p values should not be negative.

In the cell below, explain why this model has problems. That is, your above work indicates that there is something that is invalid about this model. In terms of population dynamics, what is the problem? That is, what are some assupmtions that go into the model that might not be true all the time? In what situations is the model valid?

Your answer here: An adequate correction to this model would be to include a point in the model where the population decreases. And cases where the population can change drastically. Also must get rid of the negative p (population) values.

In the cell below, use `dslove` to find the general solution to this ODE.

In [29]:
```
# Import statements
import numpy as np
import matplotlib.pyplot as plt
from sympy import *  # Needed for symbol creation

# Define variables
k = symbols('k')  # Constant
p = Function('p')  # population
t = symbols('t')  # time

# Dsolve - takes a differential equation as input,
# and the output is an equation
# deq is the differential equation
deq = Eq(p.diff(p(t), t), k*p(t))  # Eq class is What sympy uses to unders
tand equations
# essentially sp.diff(p(t), t) = k*p(t)
# diff(p(t), t) is the derivative of p(t) with repsect to t
# dsolve(deq, p(t)) says to solve deq for the variable p(t)
psoln = dsolve(deq, p(t))  # returns the list of solutions to the equation
# Good to tell dsolve exactly whta you want, but can solve most with just
dsolve(deq)
pprint(psoln)  # Pretty print = readable format
```

```
          k⋅t
p(t) = C₁⋅e
```

Another model that is a little more accurate is the following model:

$$\frac{dP}{dt} = rP(1 - kP).$$

Here, r and k are constants. In the cell below, explain how this model is different and how it (partially) corrects some of the deficiencies with the model above.

In [30]:
```
# Your answer here: This model takes into account more variables/factors
```

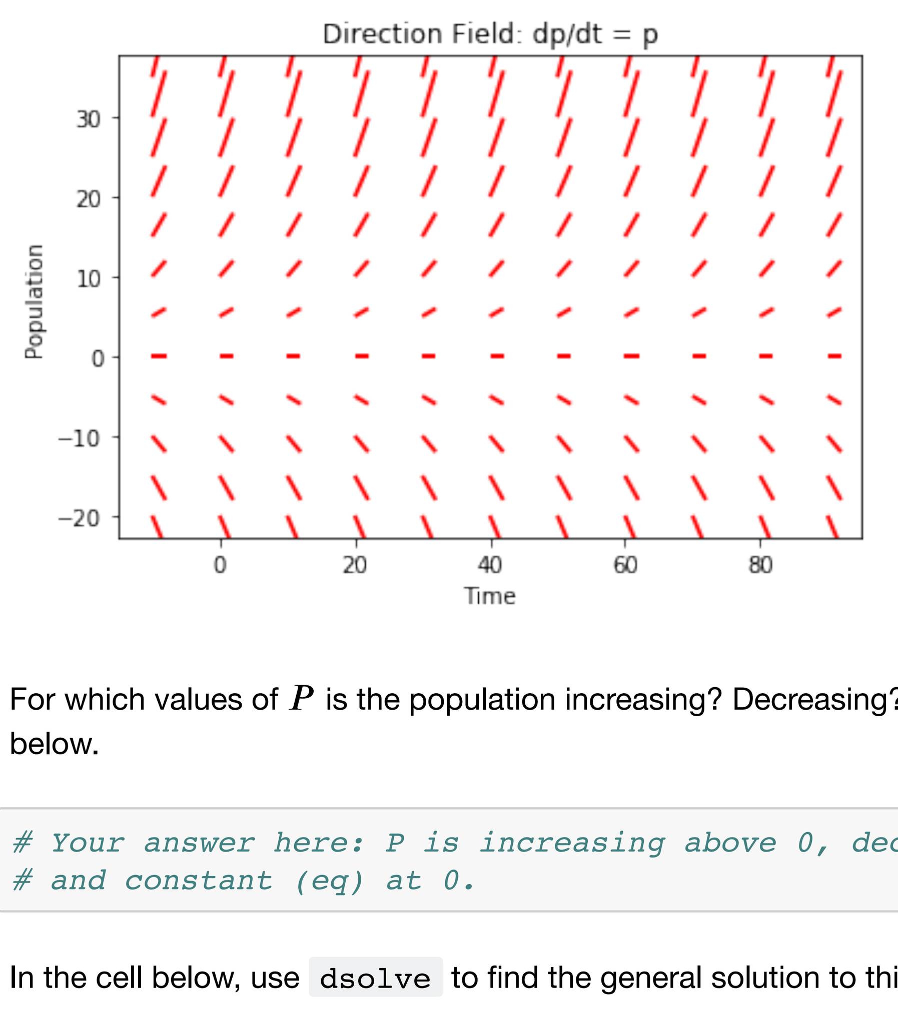
Let $k = \frac{1}{1000}$ and $r = \frac{1}{10}$. Make a direction field for this ODE in the cell below.

In [31]:
```
# Import statements
import numpy as np
import matplotlib.pyplot as plt

# Define the function
def f(t,x):
    return (1-(1/1000)*x)*x*(1/10)

# Arrays
t = np.arange(-10, 100, 10)
x = np.arange(-20, 40, 5)
T,X = np.meshgrid(t, x)
dXdT = f(T, X)
U = np.ones(T.shape) # Normalizes the arrows to see about where equilibriu
m is
V = dXdT

# Plot
plt.figure()
plt.title('Direction Field: dp/dt = p')
plt.xlabel('Time')
plt.ylabel('Population')
Q = plt.quiver(T, X, U, V,  angles='xy', headlength=0, headwidth=1, scale_
units='xy', scale=0.5, color='red')
```



For which values of P is the population increasing? Decreasing? Constant? Answer in the cell below.

In [32]:
```
# Your answer here: P is increasing above 0, decreasing below 0
# and constant (eq) at 0.
```

In the cell below, use `dsolve` to find the general solution to this ODE.

In [33]:
```
# Import statements
import numpy as np
import matplotlib.pyplot as plt
from sympy import *  # Needed for symbol creation
k = symbols('k')  # Constant
r = symbols('r') # constant
p = Function('p')  # population
t = symbols('t')  # time

# Dsolve
deq = Eq(p.diff(p(t), t), (1-k*p(t))*r*p(t))
psoln = dsolve(deq, p(t))
pprint(psoln)  # pretty print
```

```
             1
p(t) = ─────────────
         ⎛    -r⋅t    ⎞
       k⋅⎝C₁⋅e     + 1⎠
```

Grading.

This project is worth a total of 50 points. There are 10 "blanks" above. Each one is worth 5 points.

For the cells in which I ask to you explain something (these are the first, fourth, fifth, seventh, ninth blank cells above) your work will be assessed based on the following criteria:

- Is your answer written in complete sentences in the correct location? (1 pt)
- Does your response answer the question? (2 pts)
- Is your response a valid answer (that is, does it contain correct information)? (2 pts)

For the other questions, you will be graded on:

- Does your code work and produce the output that is in the document (in other words, if I were to copy a cell into a blank jupyter notebook, and run that cell only, then I should get the exact same thing that is on your pdf. This means that all import statements, etc, need to be in each cell; this is not really good programming, but is to facilitate grading.) (1 pt)
- Does your code produce the right type of output (e.g. if I asked for a direction field, is that what you gave me?) (2 pts)
- Is your output the correct output and is it formatted in a usable way (for example, if your direction field has one arrow, it isn't useful; if there are so many arrows that it just looks red, it is not useful. You can make other design decisions - e.g. you can have heads on the arrows or not; color is up to you, etc). (2 pts)

Deliverable

You will export this file as a .html file, print that as a pdf and turn it in via Gradescope.