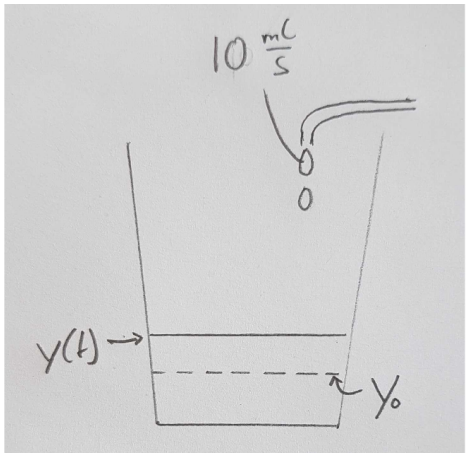
Linear and exponential growth

A simple (silly) example



Consider a bucket to which you add 10ml of water per second. The amount of water in the bucket (lets call it $y(t)$) will increase over time as

$$y(t) = y_0 + t \times 10\frac{ml}{s}$$

where y_0 is the initial amount of water. This is a simple example, but it will highlight some basic features of simple differential equations.

This very same problem can be described as "the amount of water increases by 10ml/s" which is just a differential equation:

$$\frac{dy}{dt} = 10\frac{ml}{s}$$

The right hand side is a constant -- the rate of addition -- $\alpha = 10\frac{ml}{s}$. Note that this constant has dimensions volume over time. These are the same dimensions as $\frac{dy}{dt}$ ($[y] = \text{volume}$, $[t] = \text{time}$) as it has to be. With every equation we consider, you should convince yourself that the dimensions match up!

Direct integration

We have already written down the solution of the differential equation above. More formally, we can solve this equation by integrating both sides from $0 \ldots t$:

$$\int_0^t \frac{dy}{dt'} dt' = \int_0^t \alpha dt'$$

The left hand side is

$$\int_0^t \frac{dy}{dt'} dt' = y(t) - y(0)$$

and hence simple the amount of water that was added between time 0 and time t. The right hand side is

$$\int_0^t \alpha dt' = \alpha t$$

Together with $y(0) = y_0$ we recover the solution given above.

This is of course an extremely simple example, but it will teach us fundamental properties of differential equations

- a differential equation doesn't determine the solution uniquely, we get to choose the initial condition $y(0) = y_0$
- if the differential equation is such that the right hand side does not depend on y, we can integrate it directly.

Solving differential equation with a computer

A differential equations are typically introduced as describing change in discrete little steps and then taking the limit to every smaller steps to define the differential equation. To solve differential equations with the computer, we do have to keep these steps finite and the above problem turns into a finite difference equation

$$y(t) - y(t - \Delta t) = \Delta y(t) = \frac{dy}{dt} \Delta t = \alpha \Delta t$$

For this example, we already know the answer. But learning how to solve ODEs with a computer will allow us to solve many equation to which no simple closed solution is known. It is therefore useful to explore how we would solve this equation numerically. The key idea is to rearrange the above equation as

$$y(t) = y(t - \Delta t) + \alpha \Delta t$$

We just start with $y(t) = y_0$ and add $\alpha \Delta t$ one step at a time. The computer will do this for us dutifully.

In [1]:

```
y_0 = 3  # this is a quantity with units ml
t_0 = 0  # we start at 0 seconds
y = [y_0]   # this is the list in which we will gather y(t_0), y(t_1), y(t_2), etc
t = [t_0]   # this list will contain the time points t_0, t_1, t_2

alpha = 7  # this is in ml/s

Delta_t = 0.25 # discrete time intervals
n_steps = 100  # number of steps we want to integrate, the final time will be t_0 + n_steps*Delta_t



for i in range(n_steps):
    t.append(t[i] + Delta_t)  # the new time t_[i+1] = t_[i]+Delta_t
    y.append(y[i] + alpha*Delta_t)  # the new time y_[i+1] = y_[i]+alpha*Delta_t

print("The first 10 time points:", t[:10])
print("The first 10 time values:", y[:10])
```

```
The first 10 time points: [0, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.7
5, 2.0, 2.25]
The first 10 time values: [3, 4.75, 6.5, 8.25, 10.0, 11.75, 13.5, 1
5.25, 17.0, 18.75]
```

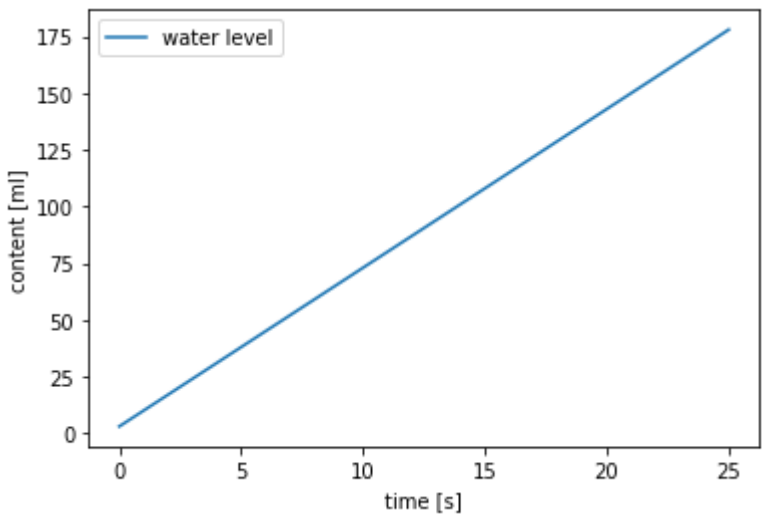
Lets graph these results!

In [3]:

```
import matplotlib.pyplot as plt

plt.plot(t, y, label='water level')
plt.xlabel('time [s]')  # label axis and specify units!
plt.ylabel('content [ml]')
plt.legend()
```

Out[3]:

```
<matplotlib.legend.Legend at 0x7f1abc560c50>
```



- change the values of `y_0`, `Delta_t`, `n_steps`, and `alpha` and see what happens.

Exponential growth

The above example is so simple that the answer is self-evident and it might have seemed silly to do all this song and dance. Now, lets consider something slightly more complicated -- for example exponential growth of bacteria.

If the bacteria divide every τ minutes and we started with a single cell, we would go from

$$1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \ldots$$

etc after a time

$$0 \rightarrow \tau \rightarrow 2\tau \rightarrow 3\tau \ldots$$

This assumes that every cells divide exactly τ minutes after is born and cells stay syncronized forever. This is probably not the case: there is a distribution of division times not a single value τ.

Alternatively, we could assume that the cells are completely desynchronized. In this case, the number of cells $n(t)$ would change in a time interval Δt approximately as

$$n(t + \Delta t) = n(t) + \frac{\Delta t}{\tau} n(t)$$

where $\frac{\Delta t}{\tau}$ is the fraction of cells that divide during the time interval Δt. This finite difference equation can be readily rearranged to resemble a differential equation

$$\lim_{\Delta t \rightarrow 0} \frac{n(t + \Delta t) - n(t)}{\Delta t} = \frac{dn(t)}{dt} = n(t)/\tau$$

This differential equation means "the rate at which $n(t)$ changes is proportional to $n(t)$" -- this is the hallmark of exponential growth. This equation is again one with an exactly known solution given by

$$n(t) = n_0 e^{t/\tau}$$

where n_0 is the initial number of cells. (Confirm this by direct differentiation).

Numerical solution

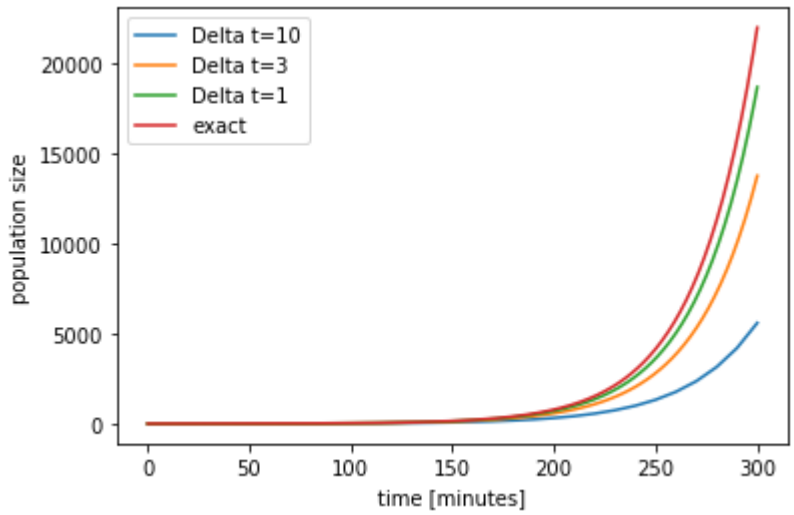
While exponential growth is again a case that as an exact solution, it is instructive to solve it numerically. We will use this example to demonstrate some challenges in numerical integration of differential equations. In particular, we will investigate how the accuracy of the solution depends on the step size Δt.

In [4]:

```
import numpy as np  # import of numerics library -- we need the exponential func
tion

tau = 30 # division time of 30 minutes
n_0 = 1
t_0 = 0
tmax = 10*tau # simulate this process for 10 times the average division time.

for Delta_t in [10,3,1]:
    n = [n_0]
    t = [t_0]
    for i in range(tmax//Delta_t):  # number of steps necessary is tmax divided
 by step size = tmax/Delta_t
        n.append(n[i] + n[i]*Delta_t/tau)
        t.append(t[i] + Delta_t)

    plt.plot(t, n, label=f"Delta t={Delta_t}")

plt.plot(t, np.exp(np.array(t)/tau), label="exact")
plt.xlabel("time [minutes]")
plt.ylabel("population size")
plt.legend()
```

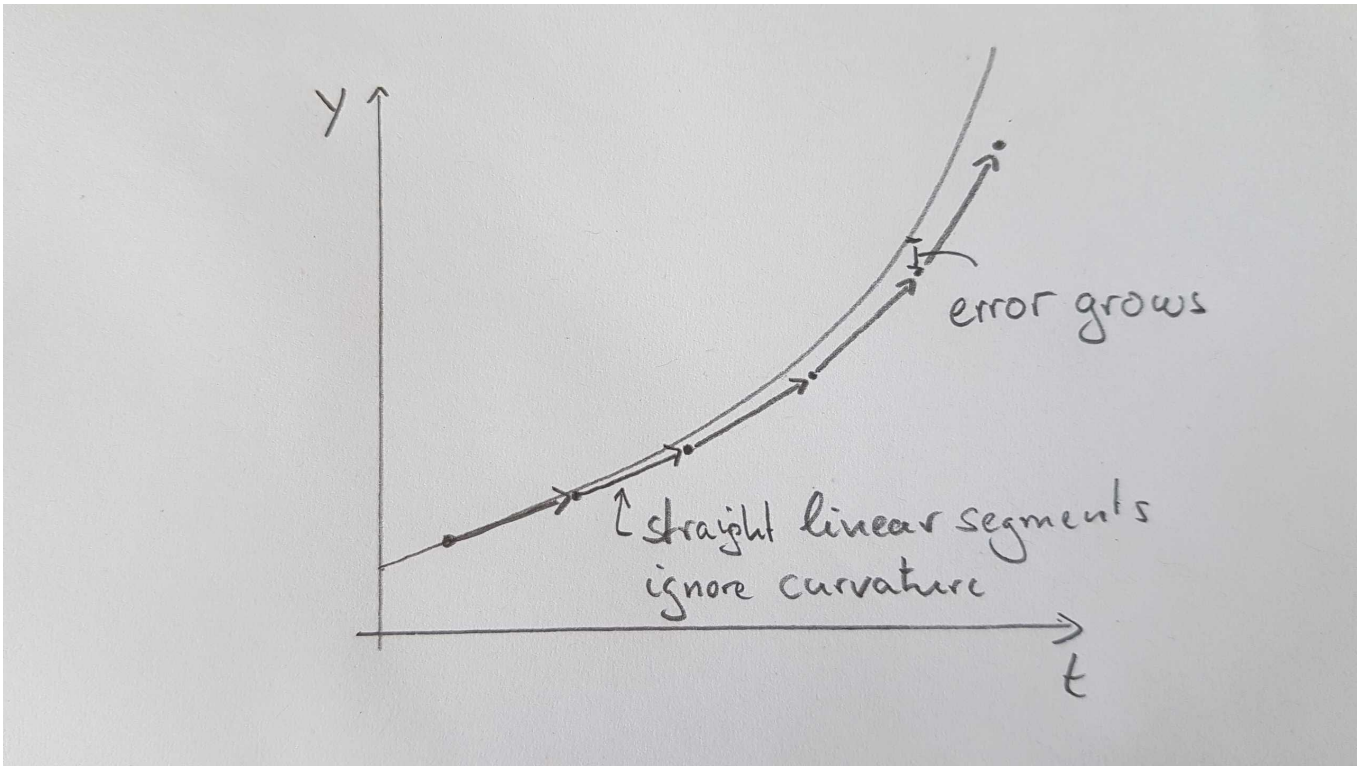
Out[4]:

```
<matplotlib.legend.Legend at 0x7f1aaf4d5650>
```



Accuracy depends on step size

As we saw above, the accuracy of the solution depends quite critically on the step size `Delta t`. The problem is that at every step, we slightly undershoot since the curve continues to bend upwards:



Sometimes, it is sufficient to simply choose a small enough step size. But more generally one needs to use a more sophisticated method than the simple forward stepping we have done here (called "Forward-Euler" method (https://en.wikipedia.org/wiki/Euler_method)). A good compromise is typically the Runge-Kutta method (https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods) which is implemented in most numerical computation packages.

For more conceptual purposes and simple exploration, the forward Euler method is still useful and we will continue to use it.

Dig deeper

- In the above example, instead of varying Δt, vary n_0 and τ. Plot the results on a linear and a logarithmic y-scale
- Solve the ODE $\frac{dn(t)}{dt} = n(t)/\tau$ using the method of Separation of variables (https://en.wikipedia.org/wiki/Separation_of_variables)
- redo the integration of the exponential growth problem using a the standard ODE integrator from scipy (https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html)

In [ ]: