Limits

You can use algebraeic methods to calculate the rate of change over a function interval by joining two points on the function with a secant line and measuring its slope. For example, a function might return the distance travelled by a cyclist in a period of time, and you can use a secant line to measure the average velocity between two points in time. However, this doesn't tell you the cyclist's vecolcity at any single point in time - just the average speed over an interval.

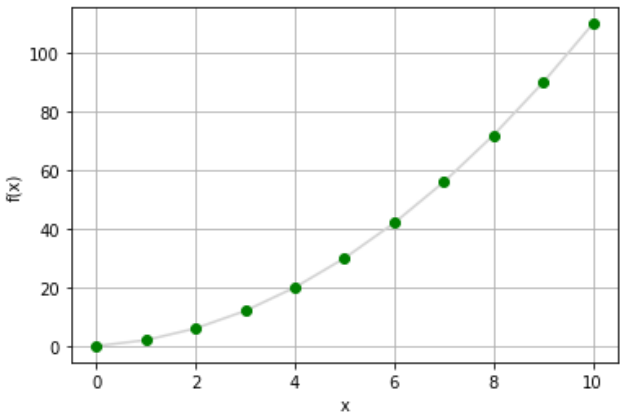
To find the cyclist's velocity at a specific point in time, you need the ability to find the slope of a curve at a given point. *Differential Calculus* enables us to do through the use of *derivatives*. We can use derivatives to find the slope at a specific x value by calculating a delta for x_1 and x_2 values that are infinitesimally close together - so you can think of it as measuring the slope of a tiny straight line that comprises part of the curve.

Introduction to Limits

However, before we can jump straight into derivatives, we need to examine another aspect of differential calculus - the *limit* of a function; which helps us measure how a function's value changes as the x_2 value approaches x_1

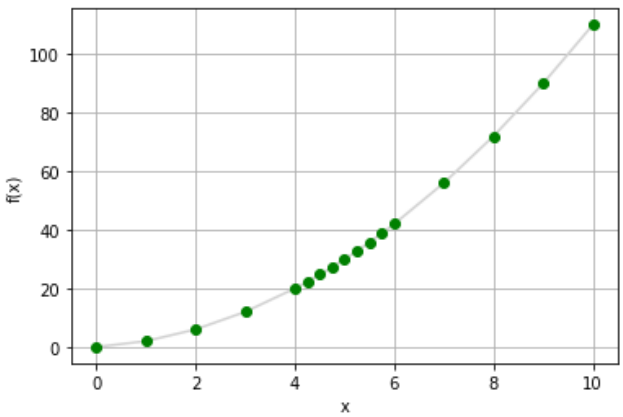
To better understand limits, let's take a closer look at our function, and note that although we graph the function as a line, it is in fact made up of individual points. Run the following cell to show the points that we've plotted for integer values of $\boldsymbol{x}$ - the line is created by interpolating the points in between:

In [1]:
```
%matplotlib inline

# Here's the function
def f(x):
    return x**2 + x

from matplotlib import pyplot as plt

# Create an array of x values from 0 to 10 to plot
x = list(range(0, 11))

# Get the corresponding y values from the function
y = [f(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid()

# Plot the function
plt.plot(x,y, color='lightgrey', marker='o', markeredgecolor='green', markerfac

plt.show()
```



We know from the function that the ***f(x)*** values are calculated by squaring the ***x*** value and adding ***x***, so we can easily calculate points in between and show them - run the following code to see this:

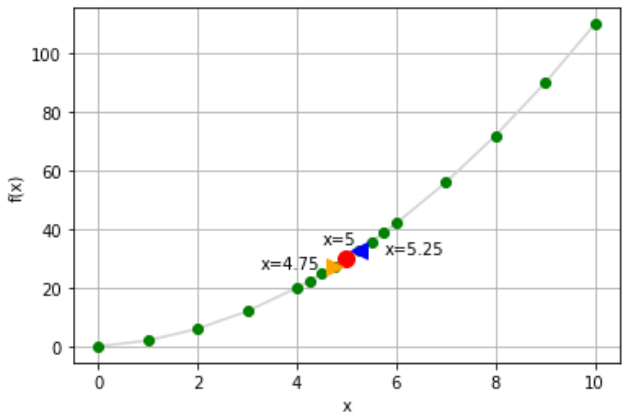
In [2]:
```
%matplotlib inline

# Here's the function
def f(x):
    return x**2 + x

from matplotlib import pyplot as plt

# Create an array of x values from 0 to 10 to plot
x = list(range(0,5))
x.append(4.25)
x.append(4.5)
x.append(4.75)
x.append(5)
x.append(5.25)
x.append(5.5)
x.append(5.75)
x = x + list(range(6,11))

# Get the corresponding y values from the function
y = [f(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid()

# Plot the function
plt.plot(x,y, color='lightgrey', marker='o', markeredgecolor='green', markerfac

plt.show()
```



Now we can see more clearly that this function line is formed of a continuous series of points, so theoretically for any given value of ***x*** there is a point on the line, and there is an adjacent point on either side with a value that is as close to ***x*** as possible, but not actually ***x***.

Run the following code to visualize a specific point for *x* = *5*, and try to identify the closest point either side of it:

In [3]:
```
%matplotlib inline

# Here's the function
def f(x):
    return x**2 + x

from matplotlib import pyplot as plt

# Create an array of x values from 0 to 10 to plot
x = list(range(0,5))
x.append(4.25)
x.append(4.5)
x.append(4.75)
x.append(5)
x.append(5.25)
x.append(5.5)
x.append(5.75)
x = x + list(range(6,11))

# Get the corresponding y values from the function
y = [f(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid()

# Plot the function
plt.plot(x,y, color='lightgrey', marker='o', markeredgecolor='green', markerfac

zx = 5
zy = f(zx)
plt.plot(zx, zy, color='red', marker='o', markersize=10)
plt.annotate('x=' + str(zx),(zx, zy), xytext=(zx - 0.5, zy + 5))

# Plot f(x) when x = 5.1
posx = 5.25
posy = f(posx)
plt.plot(posx, posy, color='blue', marker='<', markersize=10)
plt.annotate('x=' + str(posx),(posx, posy), xytext=(posx + 0.5, posy - 1))

# Plot f(x) when x = 4.9
negx = 4.75
negy = f(negx)
plt.plot(negx, negy, color='orange', marker='>', markersize=10)
plt.annotate('x=' + str(negx),(negx, negy), xytext=(negx - 1.5, negy - 1))

plt.show()
```



You can see the point where ***x*** is 5, and you can see that there are points shown on the graph that appear to be right next to this point (at *x=4.75* and *x=5.25*). However, if we zoomed in we'd see that there are still gaps that could be filled by other values of ***x*** that are even closer to 5; for example, 4.9 and 5.1, or 4.999 and 5.001. If we could zoom infinitely close to the line we'd see that no matter how close a value you use (for example, 4.999999999999), there is always a value that's fractionally closer (for example, 4.9999999999999).

So what we can say is that there is a hypothetical number that's as close as possible to our desired value of *x* without actually being *x*, but we can't express it as a real number. Instead, we express its symbolically as a *limit*, like this:

$$\lim_{x \to 5} f(x)$$

This is interpreted as *the limit of function f(x) as *x* approaches 5*.

Limits and Continuity

The function ***f(x)*** is *continuous* for all real numbered values of ***x***. Put simply, this means that you can draw the line created by the function without lifting your pen (we'll look at a more formal definition later in this course).

However, this isn't necessarily true of all functions. Consider function ***g(x)*** below:

$$g(x) = -(\frac{12}{2x})^2$$

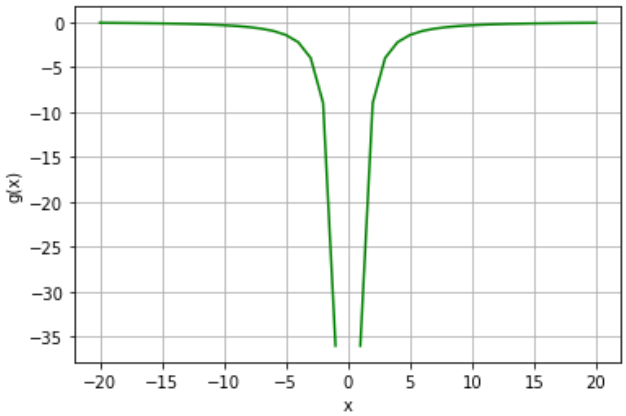
This function is a little more complex than the previous one, but the key thing to note is that it requires a division by *2x*. Now, ask yourself; what would happen if you applied this function to an *x* value of **0**?

Well, 2 • 2 is 0, and anything divided by 0 is *undefined*. So the *domain* of this function does not include 0; in other words, the function is defined when *x* is any real number such that *x is not equal to 0*. The function should therefore be written like this:

$$g(x) = -(\frac{12}{2x})^2, \quad x \neq 0$$

So why is this important? Let's investigate by running the following Python code to define the function and plot it for a set of arbitrary of values:

In [4]:
```
%matplotlib inline

# Define function g
def g(x):
    if x != 0:
        return -(12/(2*x))**2

# Plot output from function g
from matplotlib import pyplot as plt

# Create an array of x values
x = range(-20, 21)

# Get the corresponding y values from the function
y = [g(a) for a in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('g(x)')
plt.grid()

# Plot x against g(x)
plt.plot(x,y, color='green')

plt.show()
```



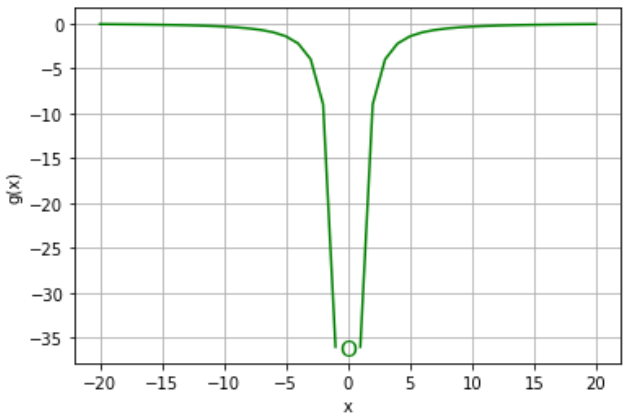
Look closely at the plot, and note the gap the line where *x* = 0. This indicates that the function is not defined here.The *domain* of the function (it's set of possible input values) not include 0, and it's *range* (the set of possible output values) does not include a value for x=0.

This is a *non-continuous* function - in other words, it includes at least one gap when plotted (so you couldn't plot it by hand without lifting your pen). Specifically, the function is non-continuous at x=0.

By convention, when a non-continuous function is plotted, the points that form a continuous line (or *interval*) are shown as a line, and the end of each line where there is a discontinuity is shown as a circle, which is filled if the value at that point is included in the line and empty if the value is not included in the line.

In this case, the function produces two intervals with a gap between them where the function is not defined, so we can show the discontinuous point as an unfilled circle - run the following code to visualize this with Python:

In [5]:
```
%matplotlib inline

# Define function g
def g(x):
    if x != 0:
        return -(12/(2*x))**2

# Plot output from function g
from matplotlib import pyplot as plt

# Create an array of x values
x = range(-20, 21)


# Get the corresponding y values from the function
y = [g(a) for a in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('g(x)')
plt.grid()

# Plot x against g(x)
plt.plot(x,y, color='green')

# plot a circle at the gap (or close enough anyway!)
xy = (0,g(1))
plt.annotate('O',xy, xytext=(-0.7, -37),fontsize=14,color='green')

plt.show()
```



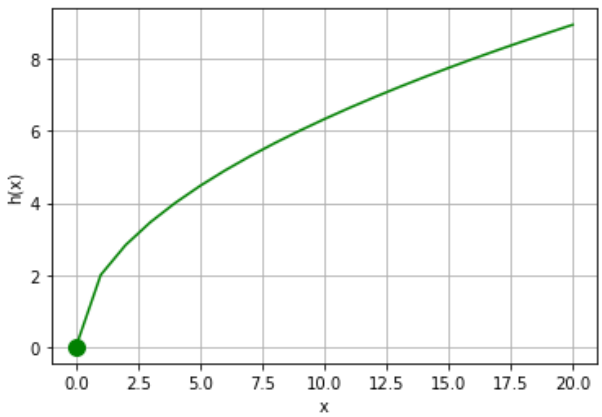
There are a number of reasons a function might be non-continuous. For example, consider the following function:

$$h(x) = 2\sqrt{x}, \quad x \geq 0$$

Applying this function to a non-negative ***x*** value returns a valid output; but for any value where ***x*** is negative, the output is undefined, because the square root of a negative value is not a real number.

Here's the Python to plot function ***h***:

In [6]:
```
%matplotlib inline

def h(x):
    if x >= 0:
        import numpy as np
        return 2 * np.sqrt(x)

# Plot output from function h
from matplotlib import pyplot as plt

# Create an array of x values
x = range(-20, 21)

# Get the corresponding y values from the function
y = [h(a) for a in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('h(x)')
plt.grid()

# Plot x against h(x)
plt.plot(x,y, color='green')

# plot a circle close enough to the h(-x) limit for our purposes!
plt.plot(0, h(0), color='green', marker='o', markerfacecolor='green', markersiz

plt.show()
```



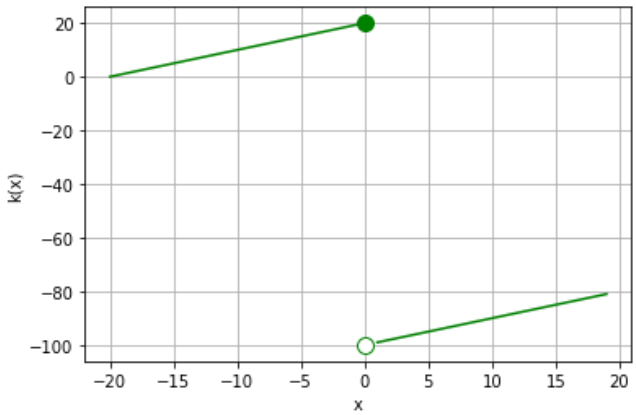
Now, suppose we have a function like this:

$$k(x) = \begin{cases} x + 20, & \text{if } x \leq 0, \\ x - 100, & \text{otherwise} \end{cases}$$

In this case, the function's domain includes all real numbers, but its output is still non-continuous because of the way different values are returned depending on the value of *x*. The *range* of possible outputs for *k(x ≤ 0)* is ≤ 20, and the range of output values for *k(x > 0)* is x > -100.

Let's use Python to plot function ***k***:

In [7]:
```
%matplotlib inline

def k(x):
    import numpy as np
    if x <= 0:
        return x + 20
    else:
        return x - 100

# Plot output from function h
from matplotlib import pyplot as plt

# Create an array of x values for each non-contonuous interval
x1 = range(-20, 1)
x2 = range(1, 20)

# Get the corresponding y values from the function
y1 = [k(i) for i in x1]
y2 = [k(i) for i in x2]

# Set up the graph
plt.xlabel('x')
plt.ylabel('k(x)')
plt.grid()

# Plot x against k(x)
plt.plot(x1,y1, color='green')
plt.plot(x2,y2, color='green')

# plot a circle at the interval ends
plt.plot(0, k(0), color='green', marker='o', markerfacecolor='green', markersiz
plt.plot(0, k(0.0001), color='green', marker='o', markerfacecolor='w', markersi

plt.show()
```



Finding Limits of Functions Graphically

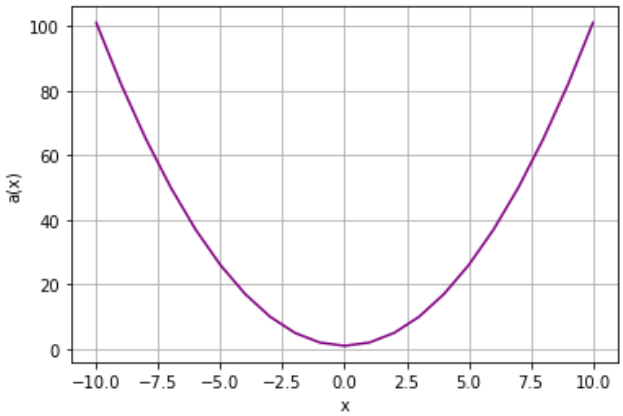
So the question arises, how do we find a value for the limit of a function at a specific point?

Let's explore this function, ***a***:

$$a(x) = x^2 + 1$$

We can start by plotting it:

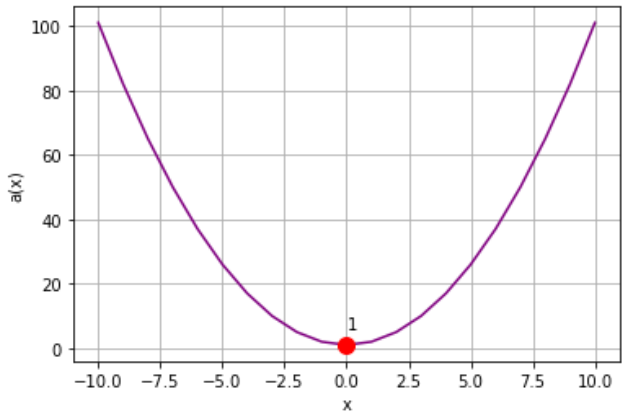
In [8]:
```
%matplotlib inline

# Define function a
def a(x):
    return x**2 + 1


# Plot output from function a
from matplotlib import pyplot as plt

# Create an array of x values
x = range(-10, 11)

# Get the corresponding y values from the function
y = [a(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('a(x)')
plt.grid()

# Plot x against a(x)
plt.plot(x,y, color='purple')

plt.show()
```



Note that this function is continuous at all points, there are no gaps in its range. However, the range of the function is *{a(x) ≥ 1}* (in other words, all real numbers that are greater than or equal to 1). For negative values of ***x***, the function appears to return ever-decreasing values as ***x*** gets closer to 0, and for positive values of ***x***, the function appears to return ever-increasing values as ***x*** gets further from 0; but it never returns 0.

Let's plot the function for an ***x*** value of 0 and find out what the ***a(0)*** value is returned:

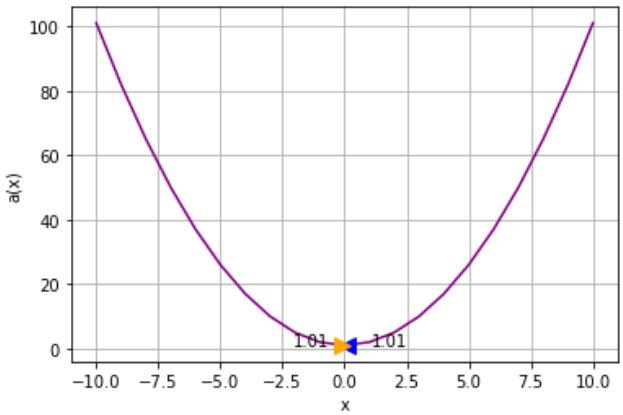
In [9]:
```
%matplotlib inline

# Define function a
def a(x):
    return x**2 + 1


# Plot output from function a
from matplotlib import pyplot as plt

# Create an array of x values
x = range(-10, 11)

# Get the corresponding y values from the function
y = [a(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('a(x)')
plt.grid()

# Plot x against a(x)
plt.plot(x,y, color='purple')

# Plot a(x) when x = 0
zx = 0
zy = a(zx)
plt.plot(zx, zy, color='red', marker='o', markersize=10)
plt.annotate(str(zy),(zx, zy), xytext=(zx, zy + 5))

plt.show()
```



OK, so ***a(0)*** returns **1**.

What happens if we use ***x*** values that are very slightly higher or lower than 0?

In [10]:
```
%matplotlib inline

# Define function a
def a(x):
    return x**2 + 1


# Plot output from function a
from matplotlib import pyplot as plt

# Create an array of x values
x = range(-10, 11)

# Get the corresponding y values from the function
y = [a(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('a(x)')
plt.grid()

# Plot x against a(x)
plt.plot(x,y, color='purple')

# Plot a(x) when x = 0.1
posx = 0.1
posy = a(posx)
plt.plot(posx, posy, color='blue', marker='<', markersize=10)
plt.annotate(str(posy),(posx, posy), xytext=(posx + 1, posy))

# Plot a(x) when x = -0.1
negx = -0.1
negy = a(negx)
plt.plot(negx, negy, color='orange', marker='>', markersize=10)
plt.annotate(str(negy),(negx, negy), xytext=(negx - 2, negy))

plt.show()
```



These ***x*** values return ***a(x)*** values that are just slightly above 1, and if we were to keep plotting numbers that are increasingly close to 0, for example 0.0000000001 or -0.0000000001, the function would still return a value that is just slightly greater than 1. The limit of function *a(x)* as *x* approaches 0, is 1; and the notation to indicate this is:

$$\lim_{x \to 0} a(x) = 1$$

This reflects a more formal definition of function continuity. Previously, we stated that a function is continuous at a point if you can draw it at that point without lifting your pen. The more mathematical definition is that a function is continuous at a point if the limit of the function as it approaches that point from both directions is equal to the function's value at that point. In this case, as we approach x = 0 from both sides, the limit is 1; and the value of *a(0)* is also 1; so the function is continuous at x = 0.

Limits at Non-Continuous Points

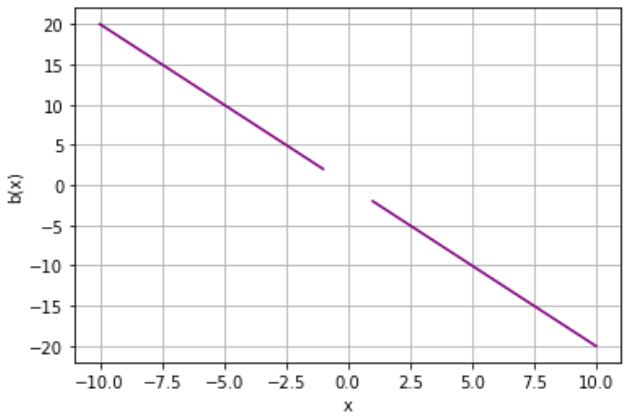
Let's try another function, which we'll call ***b***:

$$b(x) = -2x^2 \cdot \frac{1}{x}, \quad x \neq 0$$

Note that this function has a domain that includes all real number values of *x* such that *x* does not equal 0. In other words, the function will return a valid output for any number other than 0.

Let's create it and plot it with Python:

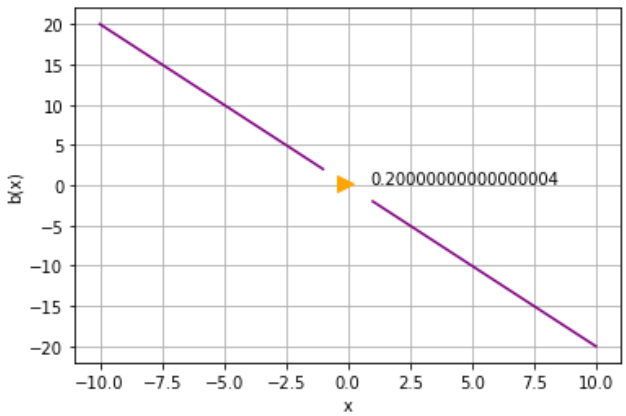
In [11]:
```
%matplotlib inline

# Define function b
def b(x):
    if x != 0:
        return (-2*x**2) * 1/x


# Plot output from function g
from matplotlib import pyplot as plt

# Create an array of x values
x = range(-10, 11)

# Get the corresponding y values from the function
y = [b(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('b(x)')
plt.grid()

# Plot x against b(x)
plt.plot(x,y, color='purple')

plt.show()
```



The output from this function contains a gap in the line where x = 0. It seems that not only does the *domain* of the function (the values that can be passed in as *x*) exclude 0; but the *range* of the function (the set of values that can be returned from it) also excludes 0.

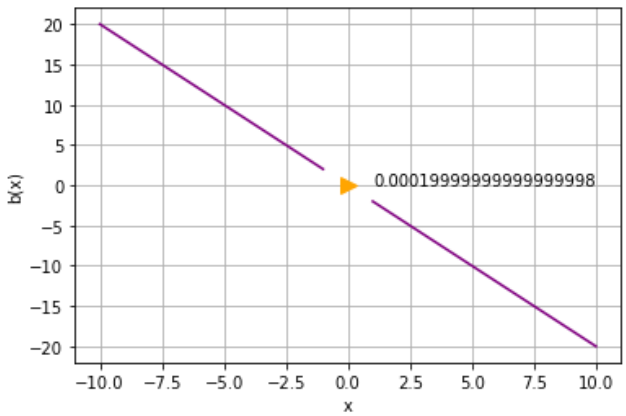
We can't evaluate the function for an *x* value of 0, but we can see what it returns for a value that is just very slightly less than 0:

In [12]:
```
%matplotlib inline

# Define function b
def b(x):
    if x != 0:
        return (-2*x**2) * 1/x


# Plot output from function g
from matplotlib import pyplot as plt

# Create an array of x values
x = range(-10, 11)

# Get the corresponding y values from the function
y = [b(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('b(x)')
plt.grid()

# Plot x against b(x)
plt.plot(x,y, color='purple')

# Plot b(x) for x = -0.1
negx = -0.1
negy = b(negx)
plt.plot(negx, negy, color='orange', marker='>', markersize=10)
plt.annotate(str(negy),(negx, negy), xytext=(negx + 1, negy))

plt.show()
```



We can even try a negative *x* value that's a little closer to 0.

In [13]:
```
%matplotlib inline

# Define function b
def b(x):
    if x != 0:
        return (-2*x**2) * 1/x


# Plot output from function g
from matplotlib import pyplot as plt

# Create an array of x values
x = range(-10, 11)

# Get the corresponding y values from the function
y = [b(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('b(x)')
plt.grid()

# Plot x against b(x)
plt.plot(x,y, color='purple')

# Plot b(x) for x = -0.0001
negx = -0.0001
negy = b(negx)
plt.plot(negx, negy, color='orange', marker='>', markersize=10)
plt.annotate(str(negy),(negx, negy), xytext=(negx + 1, negy))

plt.show()
```



So as the value of *x* gets closer to 0 from the left (negative), the value of *b(x)* is decreasing towards 0. We can show this with the following notation:

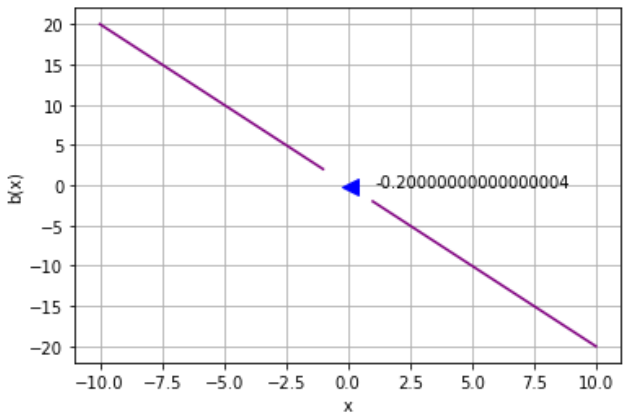
$$\lim_{x \to 0^-} b(x) = 0$$

Note that the arrow points to 0^- (with a minus sign) to indicate that we're describing the limit as we approach 0 from the negative side.

So what about the positive side?

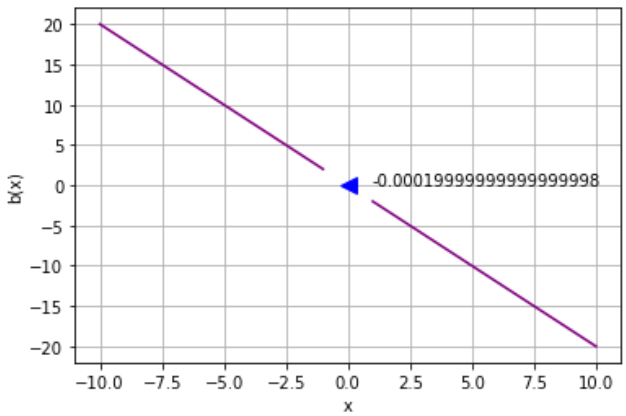
Let's see what the function value is when *x* is 0.1:

In [14]:
```
%matplotlib inline

# Define function b
def b(x):
    if x != 0:
        return (-2*x**2) * 1/x


# Plot output from function g
from matplotlib import pyplot as plt

# Create an array of x values
x = range(-10, 11)

# Get the corresponding y values from the function
y = [b(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('b(x)')
plt.grid()

# Plot x against b(x)
plt.plot(x,y, color='purple')

# Plot b(x) for x = 0.1
posx = 0.1
posy = b(posx)
plt.plot(posx, posy, color='blue', marker='<', markersize=10)
plt.annotate(str(posy),(posx, posy), xytext=(posx + 1, posy))

plt.show()
```



What happens if we decrease the value of *x* so that it's even closer to 0?

In [15]:
```
%matplotlib inline

# Define function b
def b(x):
    if x != 0:
        return (-2*x**2) * 1/x


# Plot output from function g
from matplotlib import pyplot as plt

# Create an array of x values
x = range(-10, 11)

# Get the corresponding y values from the function
y = [b(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('b(x)')
plt.grid()

# Plot x against b(x)
plt.plot(x,y, color='purple')

# Plot b(x) for x = 0.0001
posx = 0.0001
posy = b(posx)
plt.plot(posx, posy, color='blue', marker='<', markersize=10)
plt.annotate(str(posy),(posx, posy), xytext=(posx + 1, posy))

plt.show()
```



As with the negative side, as *x* approaches 0 from the positive side, the value of *b(x)* gets closer to 0; and we can show that like this:

$$\lim_{x \to 0^+} b(x) = 0$$

Now, even although the function is not defined at x = 0; since the limit as we approach x = 0 from the negative side is 0, and the limit when we approach x = 0 from the positive side is also 0; we can say that the overall, or *two-sided* limit for the function at x = 0 is 0:

$$\lim_{x \to 0} b(x) = 0$$

So can we therefore just ignore the gap and say that the function is *continuous* at x = 0? Well, recall that the formal definition for continuity is that to be continuous at a point, the function's limit as we approach the point in both directions must be equal to the function's value at that point. In this case, the two-sided limit as we approach x = 0 is 0, but *b(0)* is not defined; so the function is ***non-continuous*** at x = 0.

One-Sided Limits

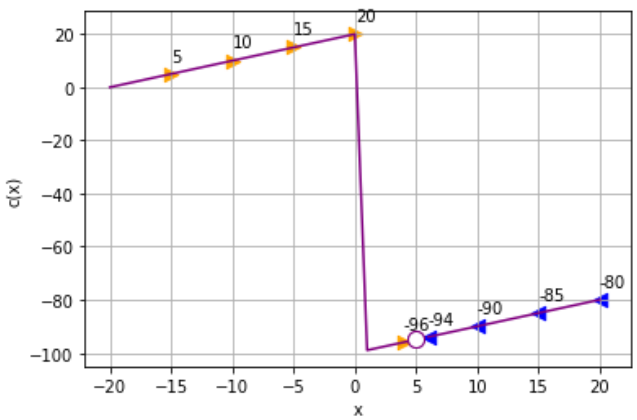
Let's take a look at a different function. We'll call this one ***c***:

$$c(x) = \begin{cases} x + 20, & \text{if } x \leq 0, \\ x - 100, & \text{otherwise} \end{cases}$$

In this case, the function's domain includes all real numbers, but its range is still non-continuous because of the way different values are returned depending on the value of *x*. The range of possible outputs for *c(x ≤ 0)* is ≤ 20, and the range of output values for *c(x > 0)* is x ≥ -100.

Let's use Python to plot function ***c*** with some values for *c(x)* marked on the line

In [16]:
```
%matplotlib inline

def c(x):
    import numpy as np
    if x <= 0:
        return x + 20
    else:
        return x - 100

# Plot output from function h
from matplotlib import pyplot as plt

# Create arrays of x values
x1 = range(-20, 6)
x2 = range(6, 21)

# Get the corresponding y values from the function
y1 = [c(i) for i in x1]
y2 = [c(i) for i in x2]

# Set up the graph
plt.xlabel('x')
plt.ylabel('c(x)')
plt.grid()

# Plot x against c(x)
plt.plot(x1,y1, color='purple')
plt.plot(x2,y2, color='purple')

# plot a circle close enough to the c limits for our purposes!
plt.plot(5, c(5), color='purple', marker='o', markerfacecolor='purple', markers
plt.plot(5, c(5.001), color='purple', marker='o', markerfacecolor='w', markersi

# plot some points from the +ve direction
posx = [20, 15, 10, 6]
posy = [c(i) for i in posx]
plt.scatter(posx, posy, color='blue', marker='<', s=70)
for p in posx:
    plt.annotate(str(c(p)),(p, c(p)),xytext=(p, c(p) + 5))

# plot some points from the -ve direction
negx = [-15, -10, -5, 0, 4]
negy = [c(i) for i in negx]
plt.scatter(negx, negy, color='orange', marker='>', s=70)
for n in negx:
    plt.annotate(str(c(n)),(n, c(n)),xytext=(n, c(n) + 5))

plt.show()
```



The plot of the function shows a line in which the *c(x)* value increases towards 25 as *x* approaches 5 from the negative side:

$$\lim_{x \to 5^-} c(x) = 25$$

However, the *c(x)* value decreases towards -95 as *x* approaches 5 from the positive side:

$$\lim_{x \to 5^+} c(x) = -95$$

So what can we say about the two-sided limit of this function at x = 5?

The limit as we approach x = 5 from the negative side is *not* equal to the limit as we approach x = 5 from the positive side, so no two-sided limit exists for this function at that point:

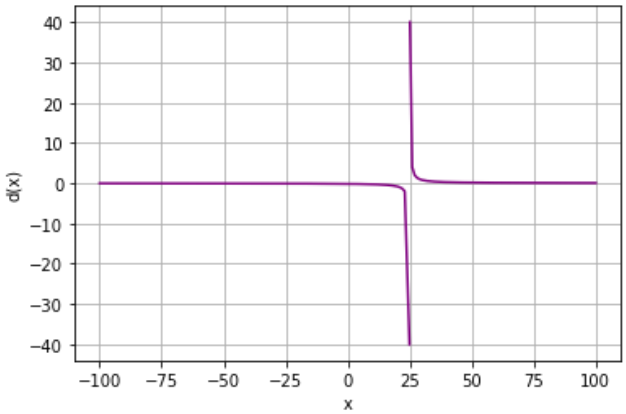
$$\lim_{x \to 5} \text{does not exist}$$

Asymptotes and Infinity

OK, time to look at another function:

$$d(x) = \frac{4}{x - 25}, \quad x \neq 25$$

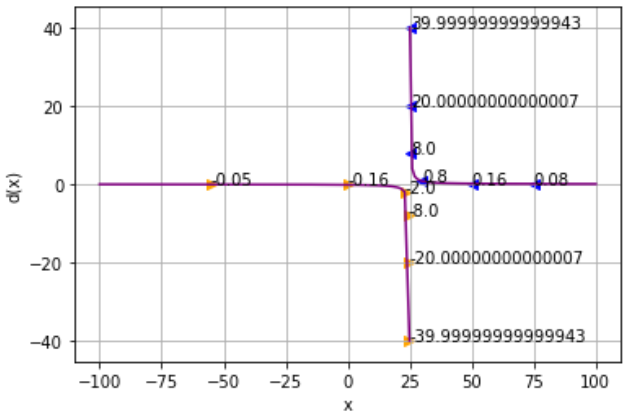
In [17]:
```
%matplotlib inline

# Define function d
def d(x):
    if x != 25:
        return 4 / (x - 25)


# Plot output from function d
from matplotlib import pyplot as plt

# Create an array of x values
x = list(range(-100, 24))
x.append(24.9) # Add some fractional x
x.append(25)   # values around
x.append(25.1) # 25 for finer-grain results
x = x + list(range(26, 101))
# Get the corresponding y values from the function
y = [d(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('d(x)')
plt.grid()

# Plot x against d(x)
plt.plot(x,y, color='purple')

plt.show()
```



What's the limit of *d* as *x* approaches 25?

We can plot a few points to help us:

In [18]:
```
%matplotlib inline

# Define function d
def d(x):
    if x != 25:
        return 4 / (x - 25)


# Plot output from function d
from matplotlib import pyplot as plt

# Create an array of x values
x = list(range(-100, 24))
x.append(24.9) # Add some fractional x
x.append(25)   # values around
x.append(25.1) # 25 for finer-grain results
x = x + list(range(26, 101))
# Get the corresponding y values from the function
y = [d(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('d(x)')
plt.grid()

# Plot x against d(x)
plt.plot(x,y, color='purple')

# plot some points from the +ve direction
posx = [75, 50, 30, 25.5, 25.2, 25.1]
posy = [d(i) for i in posx]
plt.scatter(posx, posy, color='blue', marker='<')
for p in posx:
    plt.annotate(str(d(p)),(p, d(p)))

# plot some points from the -ve direction
negx = [-55, 0, 23, 24.5, 24.8, 24.9]
negy = [d(i) for i in negx]
plt.scatter(negx, negy, color='orange', marker='>')
for n in negx:
    plt.annotate(str(d(n)),(n, d(n)))

plt.show()
```



From these plotted values, we can see that as *x* approaches 25 from the negative side, *d(x)* is decreasing, and as *x* approaches 25 from the positive side, *d(x)* is increasing. As *x* gets closer to 25, *d(x)* increases or decreases more significantly.

If we were to plot every fractional value of *d(x)* for *x* values between 24.9 and 25, we'd see a line that decreases indefintely, getting closer and closer to the x = 25 vertical line, but never actually reaching it. Similarly, plotting every *x* value between 25 and 25.1 would result in a line going up indefinitely, but always staying to the right of the vertical x = 25 line.

The x = 25 line in this case is an *asymptote* - a line to which a curve moves ever closer but never actually reaches. The positive limit for x = 25 in this case in not a real numbered value, but *infinity*:

$$\lim_{x \to 25^+} d(x) = \infty$$

Conversely, the negative limit for x = 25 is negative infinity:

$$\lim_{x \to 25^-} d(x) = -\infty$$

Finding Limits Numerically Using a Table

Up to now, we've estimated limits for a point graphically by examining a graph of a function. You can also approximate limits by creating a table of x values and the corresponding function values either side of the point for which you want to find the limits.

For example, let's return to our ***a*** function:

$$a(x) = x^2 + 1$$

If we want to find the limits as x is approaching 0, we can apply the function to some values either side of 0 and view them as a table. Here's some Python code to do that:

In [19]:
```
# Define function a
def a(x):
    return x**2 + 1


import pandas as pd

# Create a dataframe with an x column containing values either side of 0
df = pd.DataFrame ({'x': [-1, -0.5, -0.2, -0.1, -0.01, 0, 0.01, 0.1, 0.2, 0.5,

# Add an a(x) column by applying the function to x
df['a(x)'] = a(df['x'])

#Display the dataframe
df
```

Out[19]:

	x	a(x)
0	-1.00	2.0000
1	-0.50	1.2500
2	-0.20	1.0400
3	-0.10	1.0100
4	-0.01	1.0001
5	0.00	1.0000
6	0.01	1.0001
7	0.10	1.0100
8	0.20	1.0400
9	0.50	1.2500
10	1.00	2.0000

Looking at the output, you can see that the function values are getting closer to 1 as x approaches 0 from both sides, so:

$$\lim_{x \to 0} a(x) = 1$$

Additionally, you can see that the actual value of the function when x = 0 is also 1, so:

$$\lim_{x \to 0} a(x) = a(0)$$

Which according to our earlier definition, means that the function is continuous at 0.

However, you should be careful not to assume that the limit when x is approaching 0 will always be the same as the value when x = 0; even when the function is defined for x = 0.

For example, consider the following function:

$$e(x) = \begin{cases} 5, & \text{if } x = 0, \\ 1 + x^2, & \text{otherwise} \end{cases}$$

Let's see what the function returns for *x* values either side of 0 in a table:

In [20]:
```
# Define function e
def e(x):
    if x == 0:
        return 5
    else:
        return 1 + x**2

import pandas as pd
# Create a dataframe with an x column containing values either side of 0
x= [-1, -0.5, -0.2, -0.1, -0.01, 0, 0.01, 0.1, 0.2, 0.5, 1]
y =[e(i) for i in x]
df = pd.DataFrame ({' x':x, 'e(x)': y })
df
```

Out[20]:

	x	e(x)
0	-1.00	2.0000
1	-0.50	1.2500
2	-0.20	1.0400
3	-0.10	1.0100
4	-0.01	1.0001
5	0.00	5.0000
6	0.01	1.0001
7	0.10	1.0100
8	0.20	1.0400
9	0.50	1.2500
10	1.00	2.0000

As before, you can see that as the *x* values approach 0 from both sides, the value of the function gets closer to 1, so:

$$\lim_{x \to 0} e(x) = 1$$

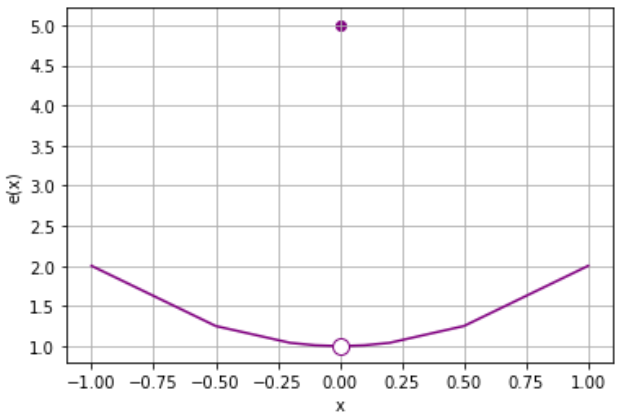
However the actual value of the function when x = 0 is 5, not 1; so:

$$\lim_{x \to 0} e(x) \neq e(0)$$

Which according to our earlier definition, means that the function is non-continuous at 0.

Run the following cell to see what this looks like as a graph:

In [21]:
```
%matplotlib inline

# Define function e
def e(x):
    if x == 0:
        return 5
    else:
        return 1 + x**2

from matplotlib import pyplot as plt

x= [-1, -0.5, -0.2, -0.1, -0.01, 0.01, 0.1, 0.2, 0.5, 1]
y =[e(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('e(x)')
plt.grid()

# Plot x against e(x)
plt.plot(x, y, color='purple')
# (we're cheating slightly - we'll manually plot the discontinous point...)
plt.scatter(0, e(0), color='purple')
# (... and overplot the gap)
plt.plot(0, 1, color='purple', marker='o', markerfacecolor='w', markersize=10)
plt.show()
```



Determining Limits Analytically

We've seen how to estimate limits visually on a graph, and by creating a table of *x* and *f(x)* values either side of a point. There are also some mathematical techniques we can use to calculate limits.

Direct Substitution

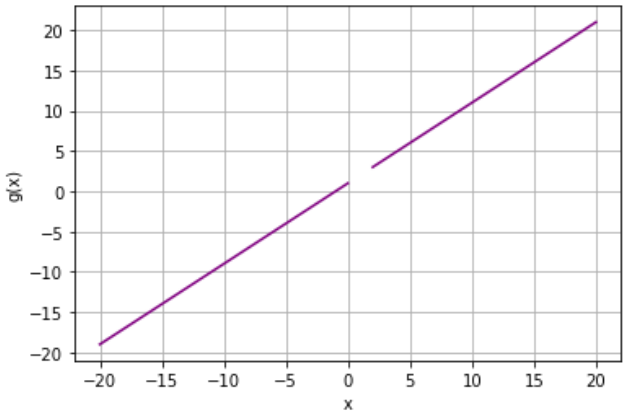
Recall that our definition for a function to be continuous at a point is that the two-directional limit must exist and that it must be equal to the function value at that point. It therefore follows, that if we know that a function is continuous at a given point, we can determine the limit simply by evaluating the function for that point.

For example, let's consider the following function ***g***:

$$g(x) = \frac{x^2 - 1}{x - 1}, x \neq 1$$

Run the following code to see this function as a graph:

In [22]:
```
%matplotlib inline

# Define function f
def g(x):
    if x != 1:
        return (x**2 - 1) / (x - 1)


# Plot output from function g
from matplotlib import pyplot as plt

# Create an array of x values
x= range(-20, 21)
y =[g(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('g(x)')
plt.grid()

# Plot x against g(x)
plt.plot(x,y, color='purple')

plt.show()
```



Now, suppose we need to find the limit of ***g(x)*** as ***x*** approaches **4**. We can try to find this by simply substituting 4 for the *x* values in the function:

$$g(4) = \frac{4^2 - 1}{4 - 1}$$

This simplifies to:

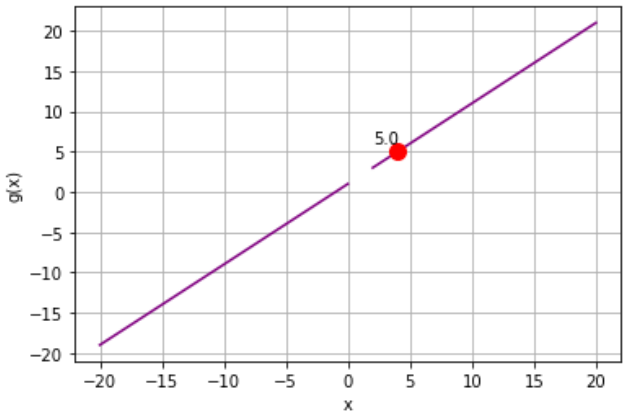
$$g(4) = \frac{15}{3}$$

So:

$$\lim_{x \to 4} g(x) = 5$$

Let's take a look:

In [23]:
```
%matplotlib inline

# Define function g
def g(x):
    if x != 1:
        return (x**2 - 1) / (x - 1)


# Plot output from function f
from matplotlib import pyplot as plt

# Create an array of x values
x= range(-20, 21)
y =[g(i) for i in x]

# Set the x point we're interested in
zx = 4

plt.xlabel('x')
plt.ylabel('g(x)')
plt.grid()

# Plot x against g(x)
plt.plot(x,y, color='purple')

# Plot g(x) when x = 0
zy = g(zx)
plt.plot(zx, zy, color='red', marker='o', markersize=10)
plt.annotate(str(zy),(zx, zy), xytext=(zx - 2, zy + 1))

plt.show()

print ('Limit as x -> ' + str(zx) + ' = ' + str(zy))
```




```
Limit as x -> 4 = 5.0
```

Factorization

OK, now let's try to find the limit of ***g(x)*** as ***x*** approaches **1**.

We know from the function definition that the function is not defined at x = 1, but we're not trying to find the *value* of ***g(x)*** when x *equals* 1; we're trying to find the *limit* of ***g(x)*** as x *approaches* 1.

The direct substitution approach won't work in this case:

$$g(1) = \frac{1^2 - 1}{1 - 1}$$

Simplifies to:

$$g(1) = \frac{0}{0}$$

Anything divided by 0 is undefined; so all we've done is to confirm that the function is not defined at this point. You might be tempted to assume that this means the limit does not exist, but $^0/_0$ is a special case; it's what's known as the *indeterminate form*; and there may be a way to solve this problem another way.

We can factor the $x^2 - 1$ numerator in the definition of ***g*** as as $(x - 1)(x + 1)$, so the limit equation can we rewritten like this:

$$\lim_{x \to a} g(x) = \frac{(x - 1)(x + 1)}{x - 1}$$

The ***x - 1*** in the numerator and the ***x - 1*** in the denominator cancel each other out:

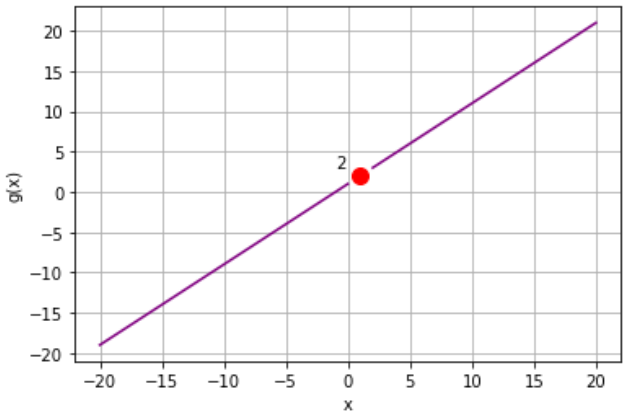
$$\lim_{x \to a} g(x) = x + 1$$

So we can now use substitution for $x = 1$ to calculate the limit as $1 + 1$:

$$\lim_{x \to 1} g(x) = 2$$

Let's see what that looks like:

In [24]:
```
%matplotlib inline

# Define function g
def f(x):
    if x != 1:
        return (x**2 - 1) / (x - 1)


# Plot output from function g
from matplotlib import pyplot as plt

# Create an array of x values
x= range(-20, 21)
y =[g(i) for i in x]

# Set the x point we're interested in
zx = 1

# Calculate the limit of g(x) when x->zx using the factored equation
zy = zx + 1

plt.xlabel('x')
plt.ylabel('g(x)')
plt.grid()

# Plot x against g(x)
plt.plot(x,y, color='purple')

# Plot the limit of g(x)
zy = zx + 1
plt.plot(zx, zy, color='red', marker='o', markersize=10)
plt.annotate(str(zy),(zx, zy), xytext=(zx - 2, zy + 1))

plt.show()

print ('Limit as x -> ' + str(zx) + ' = ' + str(zy))
```




```
Limit as x -> 1 = 2
```

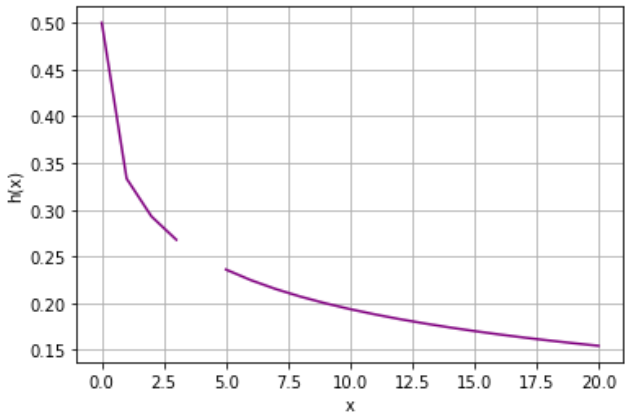
Rationalization

Let's look at another function:

$$h(x) = \frac{\sqrt{x} - 2}{x - 4}, x \neq 4 \text{ and } x \geq 0$$

Run the following cell to plot this function as a graph:

In [25]:
```
%matplotlib inline

# Define function h
def h(x):
    import math
    if x >= 0 and x != 4:
        return (math.sqrt(x) - 2) / (x - 4)


# Plot output from function h
from matplotlib import pyplot as plt

# Create an array of x values
x= range(-20, 21)
y =[h(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.ylabel('h(x)')
plt.grid()

# Plot x against h(x)
plt.plot(x,y, color='purple')

plt.show()
```



To find the limit of ***h(x)*** as ***x*** approaches **4**, we can't use the direct substitution method because the function is not defined at that point. However, we can take an alternative approach by multiplying both the numerator and denominator in the function by the *conjugate* of the numerator to *rationalize* the square root term (a conjugate is a binomial formed by reversing the sign of the second term of a binomial):

$$\lim_{x \to a} h(x) = \frac{\sqrt{x} - 2}{x - 4} \cdot \frac{\sqrt{x} + 2}{\sqrt{x} + 2}$$

This simplifies to:

$$\lim_{x \to a} h(x) = \frac{(\sqrt{x})^2 - 2^2}{(x - 4)(\sqrt{x} + 2)}$$

The $\sqrt{x^2}$ is x, and 2^2 is 4, so we can simplify the numerator as follows:

$$\lim_{x \to a} h(x) = \frac{x - 4}{(x - 4)(\sqrt{x} + 2)}$$

Now we can cancel out the *x - 4* in both the numerator and denominator:

$$\lim_{x \to a} h(x) = \frac{1}{\sqrt{x} + 2}$$

So for x approaching 4, this is:

$$\lim_{x \to 4} h(x) = \frac{1}{\sqrt{4} + 2}$$

This simplifies to:

$$\lim_{x \to 4} h(x) = \frac{1}{2 + 2}$$

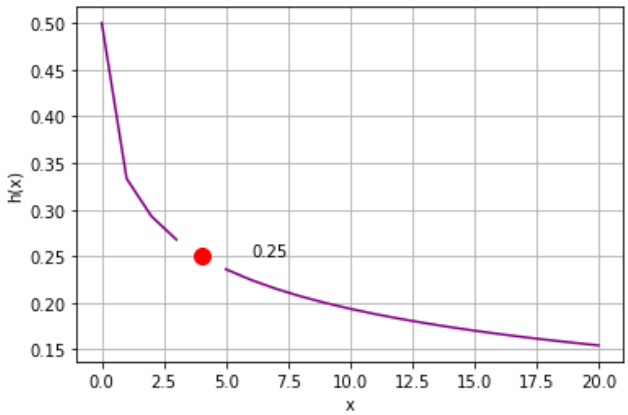
Which is of course:

$$\lim_{x \to 4} h(x) = \frac{1}{4}$$

So the limit of ***h(x)*** as ***x*** approaches **4** is $^1/_4$ or 0.25.

Let's calculate and plot this with Python:

In [26]:
```
%matplotlib inline

# Define function h
def h(x):
    import math
    if x >= 0 and x != 4:
        return (math.sqrt(x) - 2) / (x - 4)


# Plot output from function h
from matplotlib import pyplot as plt

# Create an array of x values
x= range(-20, 21)
y =[h(i) for i in x]

# Specify the point we're interested in
zx = 4

# Calculate the limit of f(x) when x->zx using factored equation
import math
zy = 1 / ((math.sqrt(zx)) + 2)

plt.xlabel('x')
plt.ylabel('h(x)')
plt.grid()

# Plot x against h(x)
plt.plot(x,y, color='purple')

# Plot the limit of h(x) when x->zx
plt.plot(zx, zy, color='red', marker='o', markersize=10)
plt.annotate(str(zy),(zx, zy), xytext=(zx + 2, zy))

plt.show()

print ('Limit as x -> ' + str(zx) + ' = ' + str(zy))
```




```
Limit as x -> 4 = 0.25
```

Rules for Limit Operations

When you are working with functions and limits, you may want to combine limits using arithmetic operations. There are some intuitive rules for doing this.

Let's define two simple functions, ***j***:

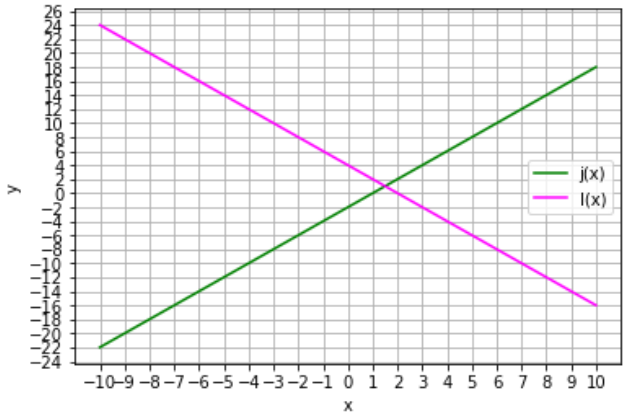
$$j(x) = 2x - 2$$

and ***l***:

$$l(x) = -2x + 4$$

Run the cell below to plot these functions:

In [27]:
```
%matplotlib inline

# Define function j
def j(x):
    return x * 2 - 2

# Define function l
def l(x):
    return -x * 2 + 4


# Plot output from functions j and l
from matplotlib import pyplot as plt

# Create an array of x values
x = range(-10, 11)

# Get the corresponding y values from the functions
jy = [j(i) for i in x]
ly = [l(i) for i in x]

# Set up the graph
plt.xlabel('x')
plt.xticks(range(-10,11, 1))
plt.ylabel('y')
plt.yticks(range(-30,30, 2))
plt.grid()

# Plot x against j(x)
plt.plot(x,jy, color='green', label='j(x)')

# Plot x against l(x)
plt.plot(x,ly, color='magenta', label='l(x)')

plt.legend()

plt.show()
```



Addition of Limits

First, let's look at the rule for addition:

$$\lim_{x \to a}(j(x) + l(x)) = \lim_{x \to a} j(x) + \lim_{x \to a} l(x)$$

What we're saying here, is that the limit of *j(x) + l(x)* as *x* approaches *a*, is the same as the limit of *j(x)* as *x* approaches *a* added to the limit of *l(x)* as *x* approaches *a*.

Looking at the graph for our functions ***j*** and ***l***, let's apply this rule to an *a* value of **8**.

By visually inspecting the graph, you can see that as *x* approaches 8 from either direction, *j(x)* gets closer to 14, so:

$$\lim_{x \to 8} j(x) = 14$$

Similarly, as *x* approaches 8 from either direction, *l(x)* gets closer to -12, so:

$$\lim_{x \to 8} l(x) = -12$$

So based on the addition rule:

$$\lim_{x \to 8}(j(x) + l(x)) = 14 + -12 = 2$$

Subtraction of Limits

Here's the rule for subtraction:

$$\lim_{x \to a}(j(x) - l(x)) = \lim_{x \to a} j(x) - \lim_{x \to a} l(x)$$

As you've probably noticed, this is consistent with the rule of addition. Based on an *a* value of 8 (and the limits we identified for this *a* value above), we can apply this rule like this:

$$\lim_{x \to 8}(j(x) - l(x)) = 14 - -12 = 26$$

Multiplication of Limits

Here's the rule for multiplication:

$$\lim_{x \to a}(j(x) \cdot l(x)) = \lim_{x \to a} j(x) \cdot \lim_{x \to a} l(x)$$

Again, you can apply this to the limits as x approached an *a* value of 8 we identified previously:

$$\lim_{x \to 8}(j(x) \cdot l(x)) = 14 \cdot -12 = -168$$

This rule also applies to multipying a limit by a constant:

$$\lim_{x \to a} c \cdot l(x) = c \cdot \lim_{x \to a} l(x)$$

So for an *a* value of 8 and a constant *c* value of 3, this equates to:

$$\lim_{x \to 8} 3 \cdot l(x) = 3 \cdot -12 = -36$$

Division of Limits

For division, assuming the limit of *l(x)* when x is approaching *a* is not 0:

$$\lim \frac{j(x)}{} = \frac{\lim_{x \to a} j(x)}{}$$