In [ ]:

```
A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

Creating a Function

In Python a function is defined using the def keyword:
```

In [1]:

```
def my_function():
  print("Hello from a function")

my_function()
```

```
Hello from a function
```

In [2]:

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

```
Emil Refsnes
Tobias Refsnes
Linus Refsnes
```

In [3]:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

```
Emil Refsnes
```

In [4]:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-49655f8043be> in <module>
      2   print(fname + " " + lname)
      3
----> 4 my_function("Emil")

TypeError: my_function() missing 1 required positional argument: 'lname'
```

In [3]:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

```
The youngest child is Linus
```

In [6]:

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child[3])

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

```
The youngest child is Linus
```

In [7]:

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

```
His last name is Refsnes
```

In [4]:

```
def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("\n\nRecursion Example Results")
tri_recursion(10)
```

```
Recursion Example Results
1
3
6
10
15
21
28
36
45
55
```

Out[4]:

```
55
```

In [9]:

```
def myfunction():
  pass

# having an empty function definition like this, would raise an error without the pass stat
```

In [10]:

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

```
15
25
45
```

In [11]:

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

```
apple
banana
cherry
```

In [12]:

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

```
I am from Sweden
I am from India
I am from Norway
I am from Brazil
```

In [ ]:

```
Python Lambda

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

Syntax

lambda arguments : expression
```

In [13]:

```
x = lambda a: a + 10
print(x(5))
```

```
15
```

In [14]:

```
x = lambda a, b: a * b
print(x(5, 6))
```

```
30
```

In [15]:

```
x = lambda a, b, c: a + b + c
print(x(5, 6, 2))
```

```
13
```

In [16]:

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

```
22
```

In [17]:

```
def myfunc(n):
  return lambda a : a * n

mytripler = myfunc(3)

print(mytripler(11))
```

```
33
```

In [18]:

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

```
22
33
```

In [ ]:

```
Python Arrays

What is an Array?
An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single

car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"
However, what if you want to loop through the cars and find a specific one? And what if you

The solution is an array!

An array can hold many values under a single name, and you can access the values by referri
```

In [ ]:

```
Array Methods

Python has a set of built-in methods that you can use on lists/arrays.

Method  Description
append()    Adds an element at the end of the list
clear() Removes all the elements from the list
copy()  Returns a copy of the list
count() Returns the number of elements with the specified value
extend()    Add the elements of a list (or any iterable), to the end of the current list
index() Returns the index of the first element with the specified value
insert()    Adds an element at the specified position
pop()   Removes the element at the specified position
remove()    Removes the first item with the specified value
reverse()   Reverses the order of the list
sort()  Sorts the list
```

In [19]:

```
cars = ["Ford", "Volvo", "BMW"]

x = cars[0]

print(x)
```

```
Ford
```

In [20]:

```
cars = ["Ford", "Volvo", "BMW"]

cars[0] = "Toyota"

print(cars)
```

```
['Toyota', 'Volvo', 'BMW']
```

In [21]:

```
cars = ["Ford", "Volvo", "BMW"]

x = len(cars)

print(x)
```

```
3
```

In [22]:

```
cars = ["Ford", "Volvo", "BMW"]

for x in cars:
  print(x)
```

```
Ford
Volvo
BMW
```

In [23]:

```
cars = ["Ford", "Volvo", "BMW"]

cars.append("Honda")

print(cars)
```

```
['Ford', 'Volvo', 'BMW', 'Honda']
```

In [24]:

```
cars = ["Ford", "Volvo", "BMW"]

cars.pop(1)

print(cars)
```

```
['Ford', 'BMW']
```

In [25]:

```
cars = ["Ford", "Volvo", "BMW"]

cars.remove("Volvo")

print(cars)
```

```
['Ford', 'BMW']
```