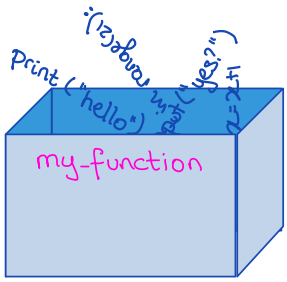
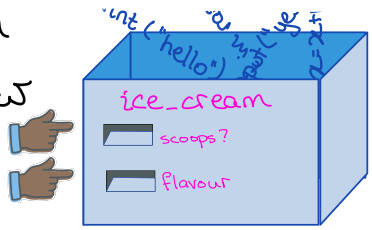
What is a function?

A function in programming is simply a bit of code that is grouped together by a name



Maybe this function needs information from me before it can do its work?

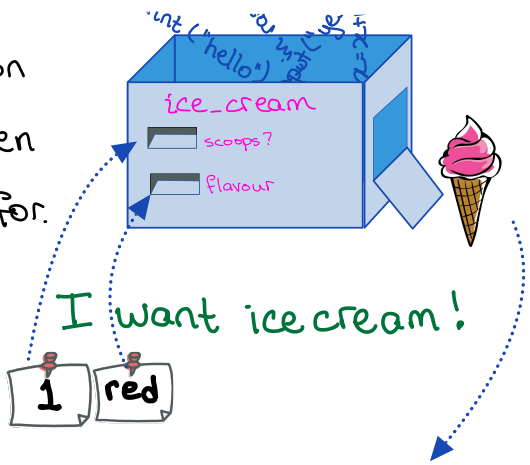
For example, to make icecream cones, we might need to know how many scoops we want and what flavour.



The individual bits of information that are required by the function are called PARAMETERS.

hey, where is my icecream!

First you have to "call" the function and feed it your choices, and then it will "return" what you asked for.



How does this translate to code?

```
def ice_cream(scoops:int, flavour:str)->cone:
    # some code ...
    return cone
treat:cone = ice_cream(1,"red")
```

NB: everything written in "green" is part of type hinting. It is good practice to use, but is not absolutely necessary.

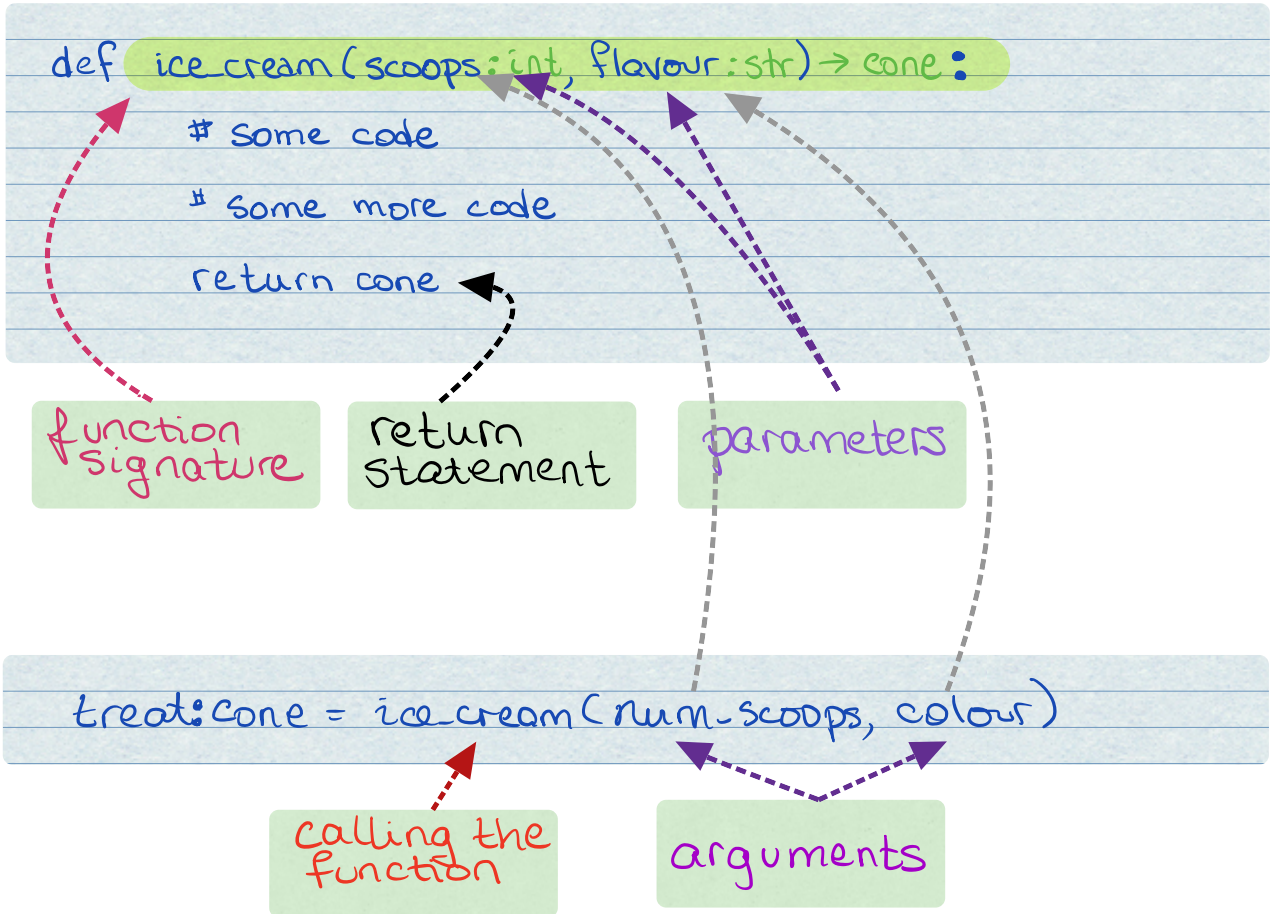
Now, lets change our program so that the user of your program can specify their own requirements.

```
def ice_cream(scoops:int, flavour:str)->cone:
    # some code ...
    return cone

num_scoops:int = int(input("how many scoops?"))
colour:str = input("what flavour colour? ")
treat:cone = ice_cream(num_scoops, colour)
```

NB: note that I am using the variables "colour" and "num_scoops" when calling "ice_cream", even though "ice_cream" uses the names "scoop" and "flavour"

Nomenclature



RULE: The number of arguments must be the same as the number of function parameters

RULE: The order of the arguments must match the order of the parameters!

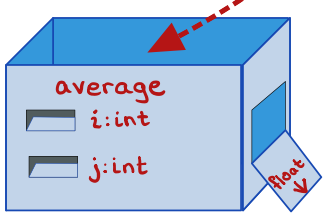
How does it work?

- Lets do a different example where the code is shown

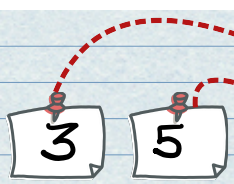
```
def average(i:int, j:int) -> float:
    ave = (i+j)/2
    return ave
```

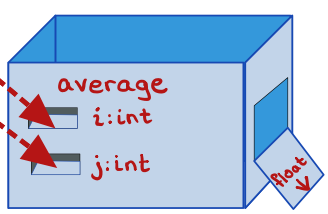
- when python sees code that starts with "def" it knows that a function is being defined. It reads all the indented code that follows.
- python then stuffs this code into an appropriate "box".

it does not execute this code!

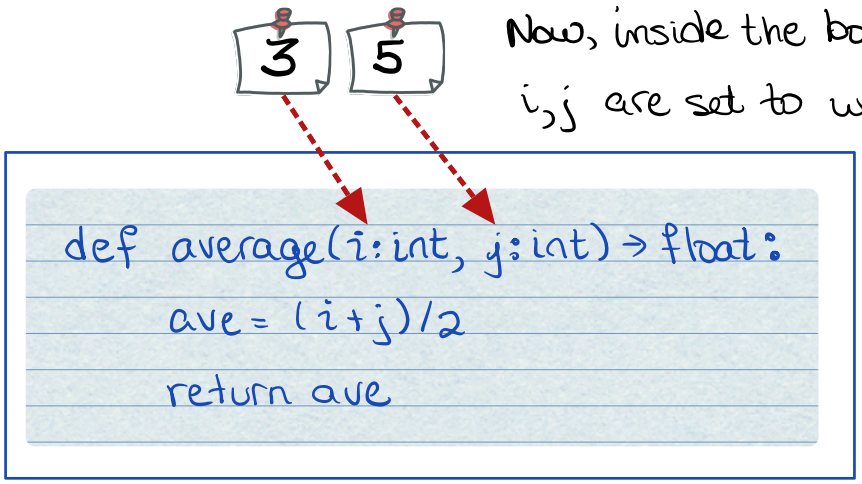



```
x=3
y=5
z=average(x,y)
```



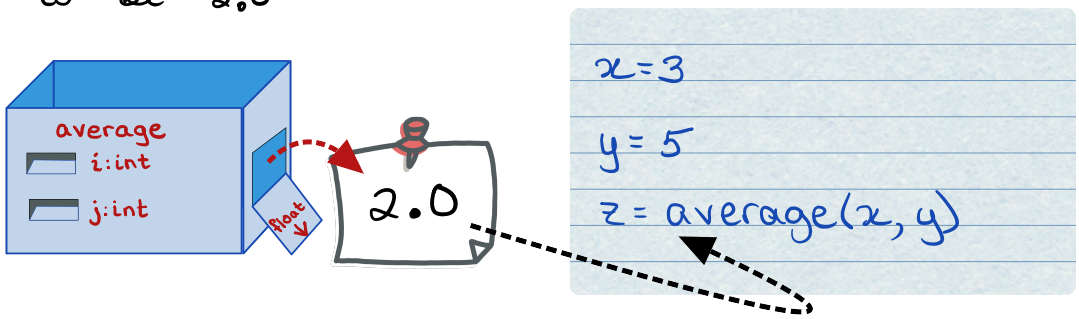


python then runs the code, and when "average" is called it puts the value of x an y into the functions parameters



Now, inside the box, the parameters i,j are set to whatever was given

i,j are now 3,5 respectively, and the average is calculated to be 2.0



The return statement returns the value to whomever called it, and the program continues running as you would expect (z is given the value of 2.0

Do functions always return a value?

NO! Sometimes you just want a function to do something, rather than calculate something.

Examples

```
print ("hi")
```

the print function writes to the console, thats all it does

```
def draw_square ( side: int):
    print ("-"*side)
    for i in range(side-2):
        inside = " "*(side-2)
        print (f"|{inside}|")
    print ("-"*side)
```

Do functions always need parameters?

NO!

Example:

You need to specify that there are no parameters

```
def help_string() -> str:
    help = "You have to remember"
    help+= "to study hard"
    return help
print ( help_string() )
```

Notice that you still need to use paranthises even if there are no arguments

Functions

Syntax:

```
def name ( parm1... ):
    code
```

} all code that is part of the function must be indented

where *name* is the name of your function.

rules: it must start with a letter or underscore (_)
- it cannot have spaces
- all characters must be either a character or a digit or an underscore
- it cannot be the same name as any other variable in your code

parms... is a list of parameters (variable names)
- there can be zero or more parameters
- each parameter must have a unique name
- each parameter name must only have characters that are letters and/or digits and/or "_"

Functions - why?

1) avoid repeating code over and over and over... Also, if we have to fix code that is repeated multiple times, then we have a higher chance of making a mistake

2) grouping code into smaller tasks, which makes it easier to read

```
def make_coffee()->drink:
    boil_water()
    prepare_machine()
    coffee=activate_machine
    return coffee
```

```
def boil_water():
    ...
def prepare_machine():
    ...
def activate_machine():
    ...
```

Can you create a function within a function?

yes you can, but DON'T!

```
def main():
    print("hello")
    def foo():
        print("bye")
    print("NO!")
```