Functions

Functions

- A `function` is a group of statements that exist within a program for the purpose of performing a specific task.

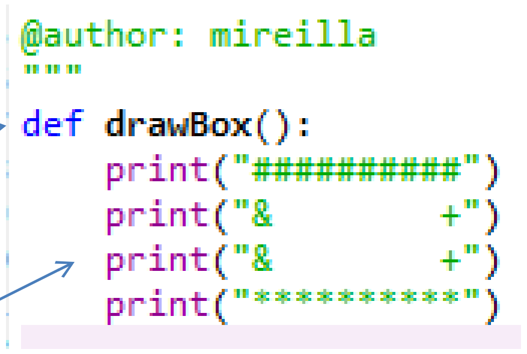
- A function is a reusable portion of a program

- We can break the programme's code into sections called *`functions`*.
- This makes it easier to develop and debug a programme as it grows.
- It can be called from many locations.

- The statements are **named** by *`defining`* a function. The statements are **executed** by *`calling`* a function.

Defining & Calling Your Functions

- A function begins with a line that consists of `def`, followed by the *`name`* of the function that is being defined, followed by an `open parenthesis`, a `close parenthesis` and a `colon`.



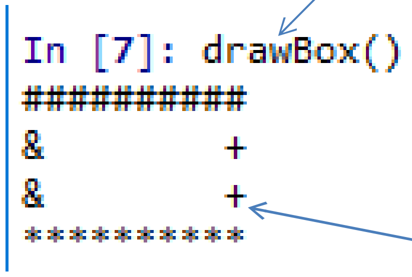
```
@author: mireilla
"""
def drawBox():
    print("###########")
    print("&         +")
    print("&         +")
    print("**********")
```

- That line is followed by the body of the function, which is a collection of statements that will execute when the function is called.

- A **function** is a set of statements that take inputs, do some specific computation and produces output.

- A function is called by its name followed by parentheses.

```
Calling the
function created
by the user
```



```
In [7]: drawBox()
###########
&         +
&         +
**********
```

```
output
```

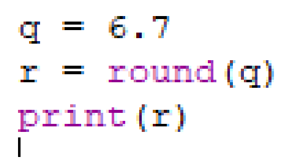
Calling Pre-defined Functions

- Many programmes perform many tasks. Some examples: reading input values from the keyboard, sorting a list, and computing the square root of a number, etc.
- Python provides **functions** that perform these common tasks. These functions have been defined by the people that created Python.

- The programme that we create will call these functions so we don't have to solve these tasks ourselves.

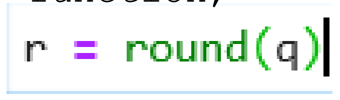
- Many functions require values when they are called such as a list of name to sort or the number for which the Square root will be computed. These values are known as **arguments** and are placed inside the parentheses when the function is called.

- A function call can have many arguments separated by a comma.



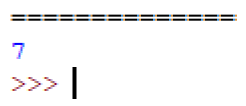
```
q = 6.7
r = round(q)
print(r)
```

Example: This assignment statement calls the **round** function,



```
r = round(q)
```

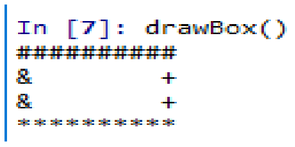
which rounds a number to the nearest integer. When it is run, the output is:



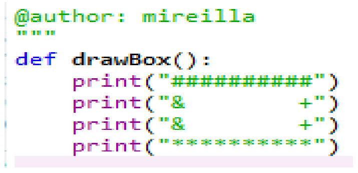
```
==============
7
>>> 
```

Functions with Parameters

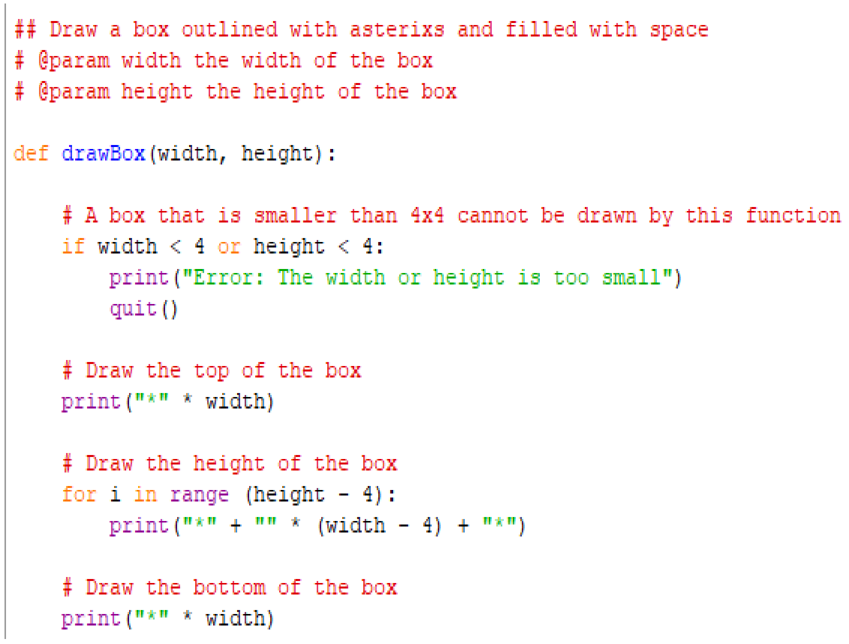
- Our basic example of function, drawBox works correctly but it is not flexible, so not very useful.
- For it to be useful and flexible, we want our function to draw boxes of many different sizes.
- Our function will take *arguments* (see previous slide). It will receive these argument values in *parameter variables* that are included inside the parentheses when the function is defined.
- In our example we will add two parameters to the definition of drawBox which contains the width and height of the box.



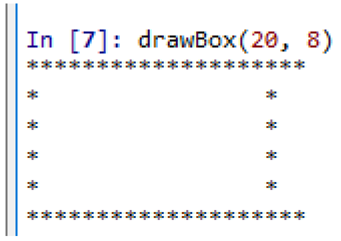
```
In [7]: drawBox()
##########
&        +
&        +
**********
```



```
@author: mireilla
"""
def drawBox():
    print("##########")
    print("&        +")
    print("&        +")
    print("**********")
```



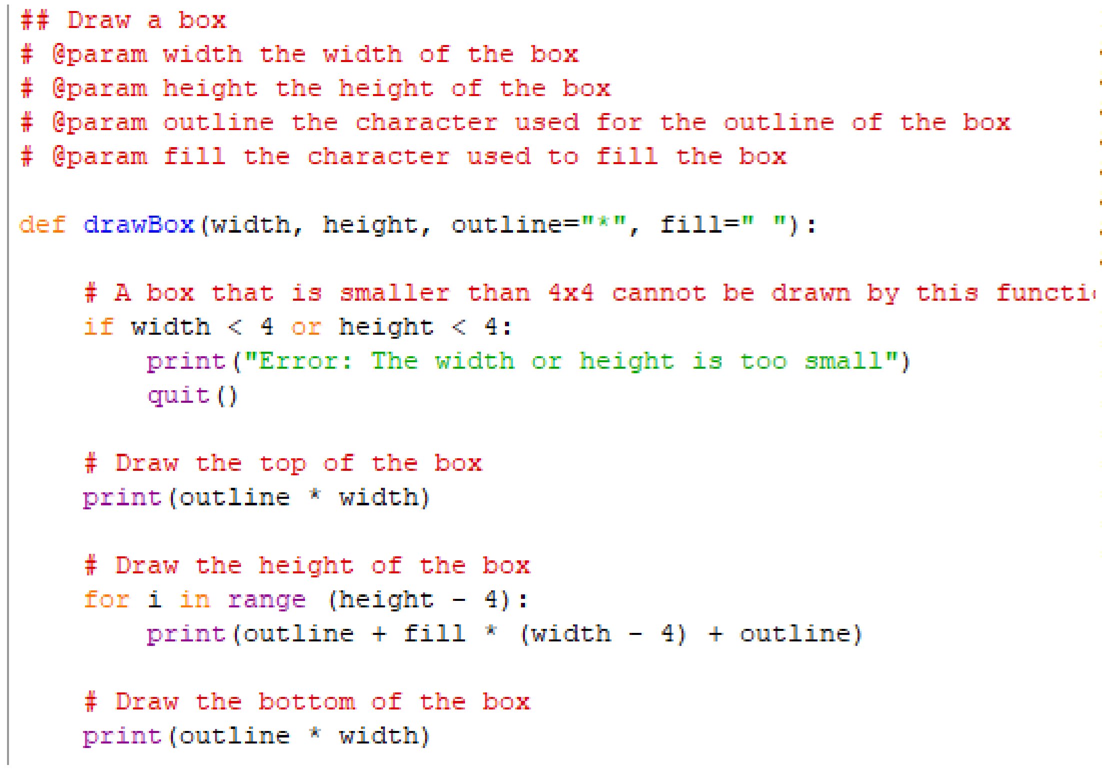
```
## Draw a box outlined with asterixs and filled with space
# @param width the width of the box
# @param height the height of the box

def drawBox(width, height):

    # A box that is smaller than 4x4 cannot be drawn by this function
    if width < 4 or height < 4:
        print("Error: The width or height is too small")
        quit()

    # Draw the top of the box
    print("*" * width)

    # Draw the height of the box
    for i in range (height - 4):
        print("*" + "" * (width - 4) + "*")

    # Draw the bottom of the box
    print("*" * width)
```



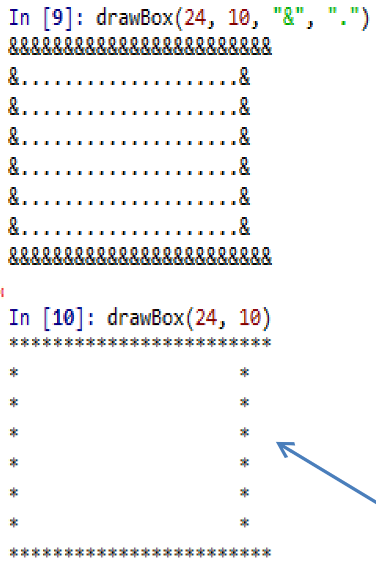
```
In [7]: drawBox(20, 8)
********************
*                  *
*                  *
*                  *
*                  *
********************
```

Functions with Parameters

- Now we want to update our function to enable the user to change the outline (previously asterix) with a different character and fill it with different characters. We add two additional parameters: `outline` and `fill`



```python
## Draw a box
# @param width the width of the box
# @param height the height of the box
# @param outline the character used for the outline of the box
# @param fill the character used to fill the box

def drawBox(width, height, outline="*", fill=" "):

    # A box that is smaller than 4x4 cannot be drawn by this functi
    if width < 4 or height < 4:
        print("Error: The width or height is too small")
        quit()

    # Draw the top of the box
    print(outline * width)

    # Draw the height of the box
    for i in range (height - 4):
        print(outline + fill * (width - 4) + outline)

    # Draw the bottom of the box
    print(outline * width)
```



```
In [9]: drawBox(24, 10, "&", ".")
&&&&&&&&&&&&&&&&&&&&&&&&
&....................&
&....................&
&....................&
&....................&
&....................&
&....................&
&&&&&&&&&&&&&&&&&&&&&&&&

In [10]: drawBox(24, 10)
************************
*                    *
*                    *
*                    *
*                    *
*                    *
*                    *
************************
```

Output when the function call includes all 4 arguments

Output when the function call includes the first 2 arguments. The default ones asterisk and empty space are used for 3rd and 4th argument

Variables in Functions

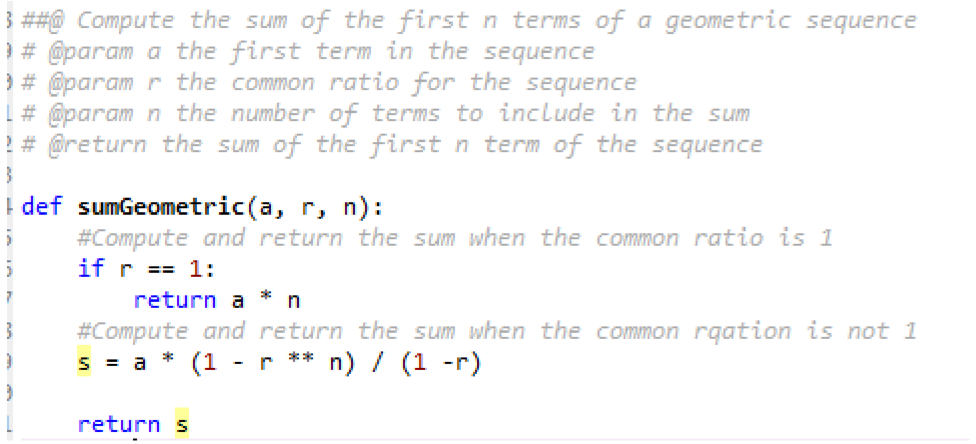
- `Local variable`: a variable is created inside a function. Only exists when the function is executing and can only be accessed within the body of that function. Cannot be access when the function returns.

- Variables created with assignment statements in the body of a function are also local variables.

- The drawBox function uses several variables.
  - Parameter variable such as `width` and `fill` that are created when the function is called
  - The `for` loop control variable, `i`, that is created when the loops begins to execute.

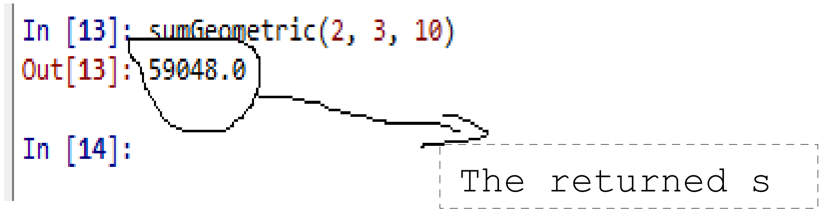
Return Values

- Our drawBox function only outputs characters on the screen.
- Many functions can take an argument, compute a value that is stored in a variable and used later in the programme.



```
##@ Compute the sum of the first n terms of a geometric sequence
# @param a the first term in the sequence
# @param r the common ratio for the sequence
# @param n the number of terms to include in the sum
# @return the sum of the first n term of the sequence

def sumGeometric(a, r, n):
    #Compute and return the sum when the common ratio is 1
    if r == 1:
        return a * n
    #Compute and return the sum when the common rqation is not 1
    s = a * (1 - r ** n) / (1 -r)

    return s
```

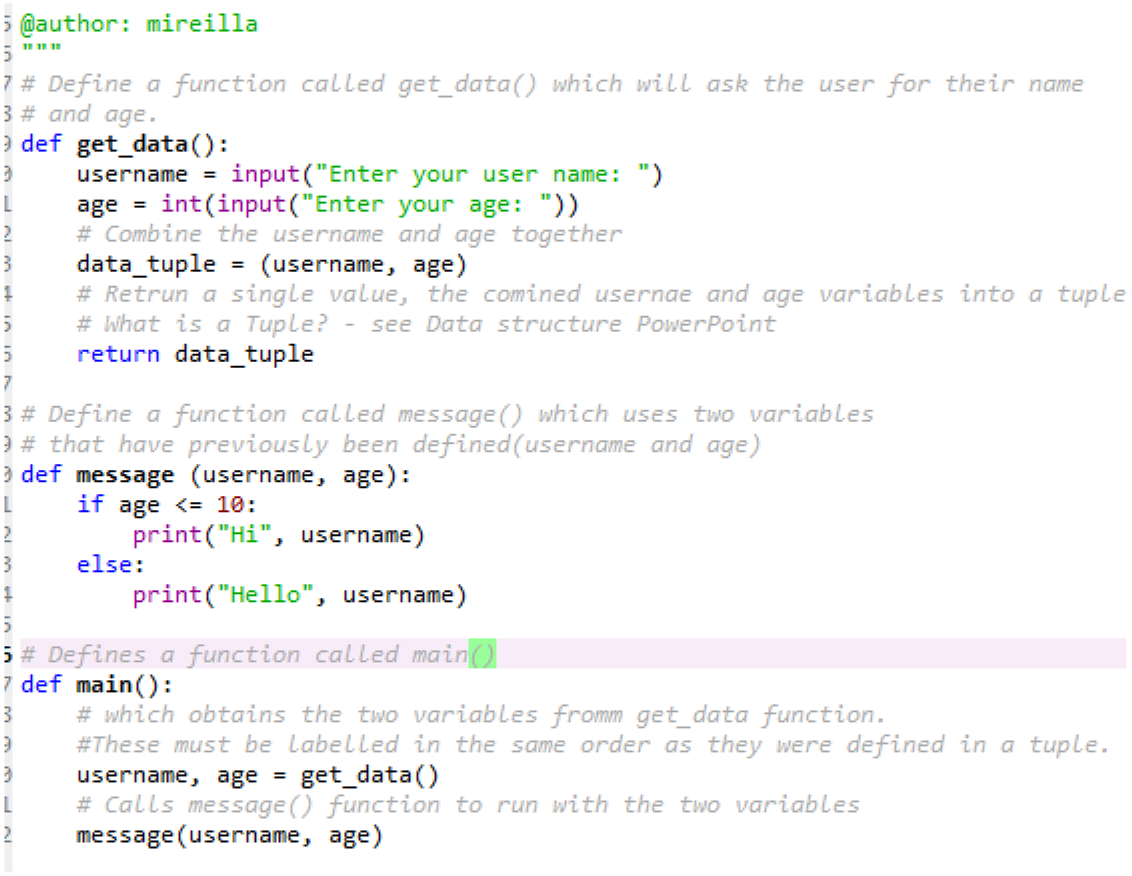
- For example the `input` function reads value typed by the user and then returns it so that it can be used later in the programme. Similarly, the `sqrt` function in the math module computes the square root of its argument and returns this value that will be used in other calculation



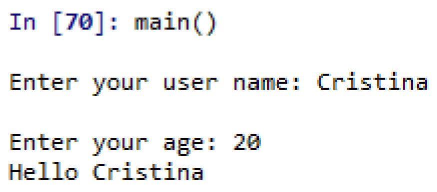
```
In [13]: sumGeometric(2, 3, 10)
Out[13]: 59048.0

In [14]:
```

- A function returns a value using a `return` function

Function code example



```python
@author: mireilla
"""
# Define a function called get_data() which will ask the user for their name
# and age.
def get_data():
    username = input("Enter your user name: ")
    age = int(input("Enter your age: "))
    # Combine the username and age together
    data_tuple = (username, age)
    # Retrun a single value, the comined usernae and age variables into a tuple
    # What is a Tuple? - see Data structure PowerPoint
    return data_tuple

# Define a function called message() which uses two variables
# that have previously been defined(username and age)
def message (username, age):
    if age <= 10:
        print("Hi", username)
    else:
        print("Hello", username)

# Defines a function called main()
def main():
    # which obtains the two variables fromm get_data function.
    #These must be labelled in the same order as they were defined in a tuple.
    username, age = get_data()
    # Calls message() function to run with the two variables
    message(username, age)
```



```
In [70]: main()

Enter your user name: Cristina

Enter your age: 20
Hello Cristina
```

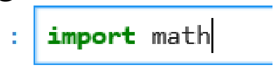
Input and Output Functions

- **Read input**
- Python reads input from keyboard by calling the `input()` function.
- `a = input():` A value type by the user will be stored in the variable called a.
- The input function always returns a string

- **Display output**
- `print()` function. Can be called with one argument.
- Can print multiple values.
- Arguments to a function call can be values and variables.

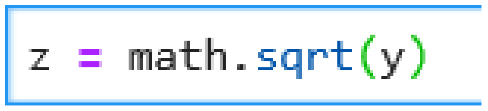
Importing Functions into Other Programmes

- **Import function**
- Functions like input and print are used in many programmes and are available in those programmes. The less commonly used function are stored in libraries/modules and need to be imported when neede



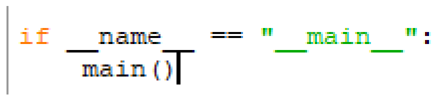
```
import math
```

- For example, additional mathematical functions are stored in the `math` module. Functions in `math` module include `sqrt`, `ceil` and `sin`.
- A function imported from a module is called by using the module name, followed by a period, followed by the name of the function and its arguments.



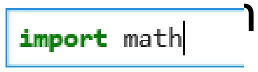
```
z = math.sqrt(y)
```

- A function call from different locations is easy when the function definition and call locations are in the same file.
- If you want to call a function that you wrote for a previous programme while solving a new problem, use the `import` keyword, followed by the name of the Python file that contains the function you want (without `.py` extension). This calls all those functions but also run those programme.
- If we only want to call those functions without running the programme, create a function called `main` that will contains the statements needed to solve the problems. Add this line of code to ensure that the `main` function does not execute when the file has been imported into another programme.



```
if __name__ == "__main__":
    main()
```

Maths with Python

As mentioned in the previous slide, Python can perform several mathematical functions. Data is either integer (a whole number) or floating-point number (number with a decimal place). In order to use some of the mathematical functions (e.g. `math.sqrt.(` 

```
import math
```

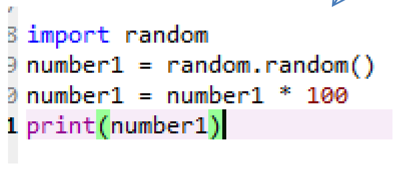
nd `math.pi`), you will need to import the maths library.

- `print(round(num, 2))` displays the number rounded to two decimals
- `**` means to the power of (e.g. 10^2 is `10**2`).
- `math.sqrt(number1)`: The square root of a number
- `number1 = float(input(“Enter number: “))` Allows numbers with a decimal point dividing the integer and fraction part.
- `math.pi` gives the pi to 15 decimal places
- `x // y`: Whole number division (e.g. `15//2` gives the answer `7`
- `x % y`: Finds the remainder (e.g. `15%2` gives the answer `1`)

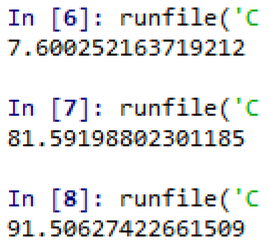
Random

- You can generate random values in Python using `random` library.
- These random values can be:
  - Random numbers within a specified range.
  - A random choice from a range of items that are input.

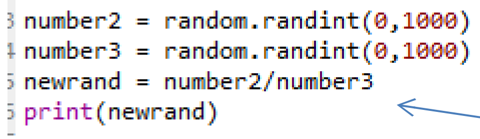
`number1 = random.random` will select a random floating-point number between 0 and 1 and stores it in a variable called number1. It is multiplied by 100 to get a larger number



```
import random
number1 = random.random()
number1 = number1 * 100
print(number1)
```



```
In [6]: runfile('C
7.600252163719212

In [7]: runfile('C
81.59198802301185

In [8]: runfile('C
91.50627422661509
```

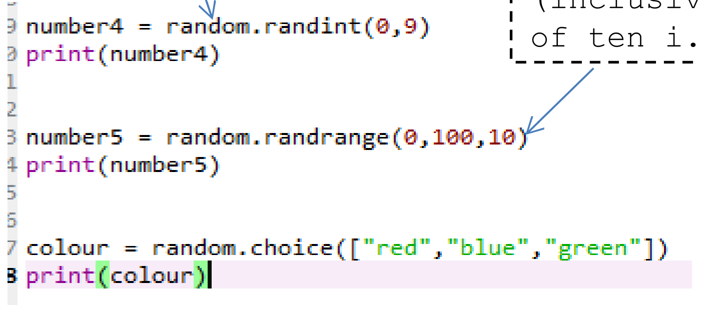


```
number2 = random.randint(0,1000)
number3 = random.randint(0,1000)
newrand = number2/number3
print(newrand)
```

Creates a random floating-point number by creating two random integers within two large range and dividing one by another

Number4: Selects a random whole number between 0 and 9 (inclusive)

Pick a random number between 0 and 100 (inclusive) in steps of ten i.e. 0,10,20,

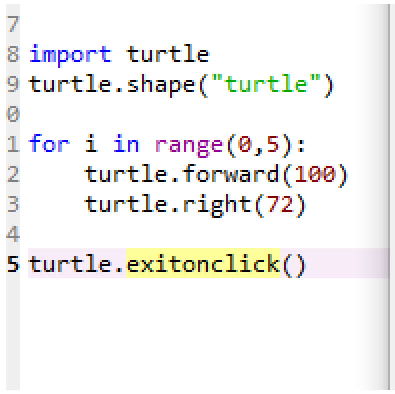


```
number4 = random.randint(0,9)
print(number4)


number5 = random.randrange(0,100,10)
print(number5)


colour = random.choice(["red","blue","green"])
print(colour)
```

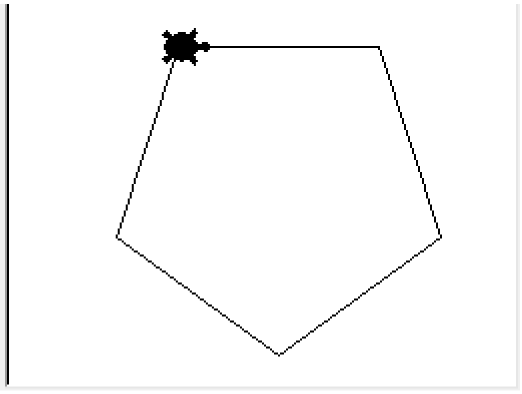
Picks a random value from the options "red", "blue" and "green" and stores it as the variable "colour"

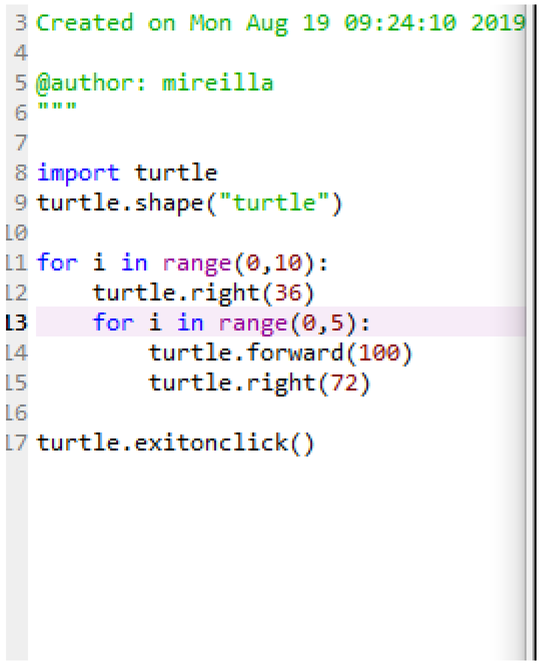
Turtle Graphics

- You can draw intricate shapes using `turtle` in Python using programmes that repeat simple moves such as loops.
- As seen in the example, a pentagon is drawn using the code next to it.

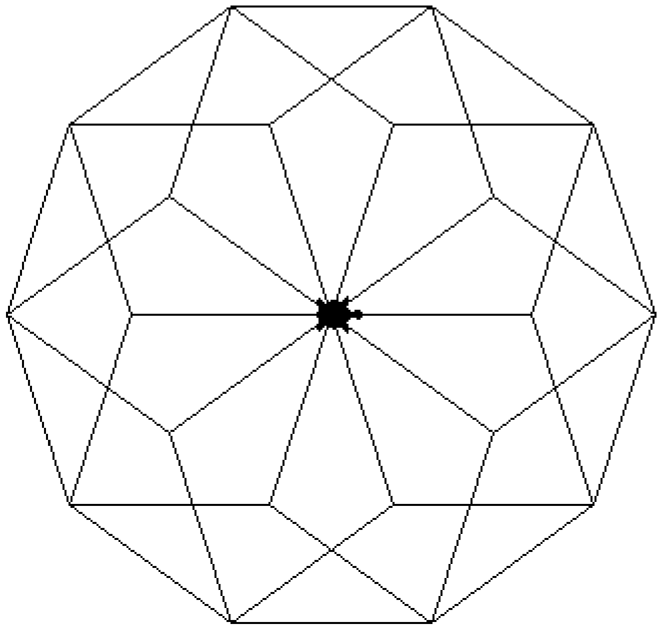


```python
import turtle
turtle.shape("turtle")

for i in range(0,5):
    turtle.forward(100)
    turtle.right(72)

turtle.exitonclick()
```





```python
Created on Mon Aug 19 09:24:10 2019

@author: mireilla
"""

import turtle
turtle.shape("turtle")

for i in range(0,10):
    turtle.right(36)
    for i in range(0,5):
        turtle.forward(100)
        turtle.right(72)

turtle.exitonclick()
```



Using `nested` loops (a loop inside a loop) a beautiful pattern is created.