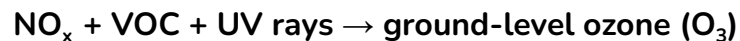


Lesson 6 - Functions

Definition: a block of code that only runs when it is called

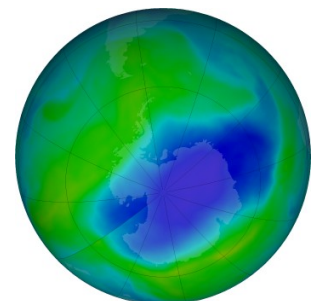
Smog & Ozone

Smog, a term created by mixing the words smoke and fog, originally described the burning of coal. Today, the term smog is used to describe photochemical smog, a reaction of the sun's UV rays, nitrogen oxides (NO_x) and volatile organic compounds (VOC's). Nitrogen oxides, a group of reactive gases such as NO₂ and NO, are created through the combustion of fuel in vehicles, power plants and factories. VOCs are released into the atmosphere from cleaning solvents and paints, amongst other products. When NO_x and VOCs react with the sun's UV rays, they form smog, which is largely composed of ground level ozone (O₃).



Ground-level ozone, located in the troposphere, has many adverse health effects including a variety of respiratory effects that impact lung function and harm lung tissue. Ozone located in the upper layer of the stratosphere however, 15km/9.3 miles to 30 km/18.6 miles from the surface of the Earth, serves a protective function as it absorbs the sun's UV rays. So whether ozone is helpful or harmful is completely dependent on where it is found. Though ozone is only a trace gas in the stratosphere, it's absorption of the sun's UV rays is important as UV rays can damage the outer layers, like skin, on living organisms and cause skin cancer (melanoma).

The ozone layer, however, is thinning due to chemicals called chlorofluorocarbons (CFC's), which can be found in refrigerants and plastic products. The "ozone hole", a term commonly used to describe patches of the layer that are particularly thin, occurs primarily in the poles, with the largest patch located over Antarctica reaching a size of 24.8 million km² / 9.6 million miles² (3 times the size of continental USA) on September 20, 2020. [Photo source: NASA](#)



Sometimes it is preferable to store data as **functions** rather than simply printing data, so that data can be **returned** (called) later. When defining a function, **parameters** (also referred to as **arguments**) are put in the brackets beside it. This indicates how many arguments, if any (it can also be empty), are passed into this function.

Part 1: Define and create a function

Let's say that it's a particularly smoggy day due to a lot of hours of sunlight, temperature inversions (when warm air doesn't rise) and traffic, a lot of tropospheric ozone is created. As typical healthy levels are around 54 ppb (parts per billion), the conditions on this particular day have doubled the ground level ozone.

1. To create a function, define it using **def** followed by the **name** of the function (in this case: **bad_conditions**), **parenthesis ()** with the argument **ozone** and a **colon :**
`def bad_conditions(ozone):`
2. **Indent** the following line, and create a **return** statement that indicates that ozone has been **multiplied by 2**.
`return 2 * ozone`
3. **Print** the variable **bad_conditions** with the value of 54.

Example:

```
main.py
1 def bad_conditions(ozone):
2     return 2 * ozone
3 print (bad_conditions(54))
```

108

In this example there is only one argument: ozone, which will be multiplied by 2 to indicate the doubling of the ozone in the atmosphere. To pass a specific value into this function, 54 ppb, it is placed in the same position as the argument for this function.

Part 2: Define a function with multiple arguments

Nitrogen dioxide (NO₂) is part of a group of highly reactive gases called nitrogen oxides (NO_x) and is a major contributor to smog. It is formed by the burning of fuel in transportation, power plants, and industrial processes.

1. Define a function for nitrogen dioxide with two **arguments** (information that is passed through a function) inside the parentheses, separated by a comma: `pollutant1` and `pollutant2` and a **colon** at the end of the statement.

```
def NO2 (pollutant1, pollutant2):
```

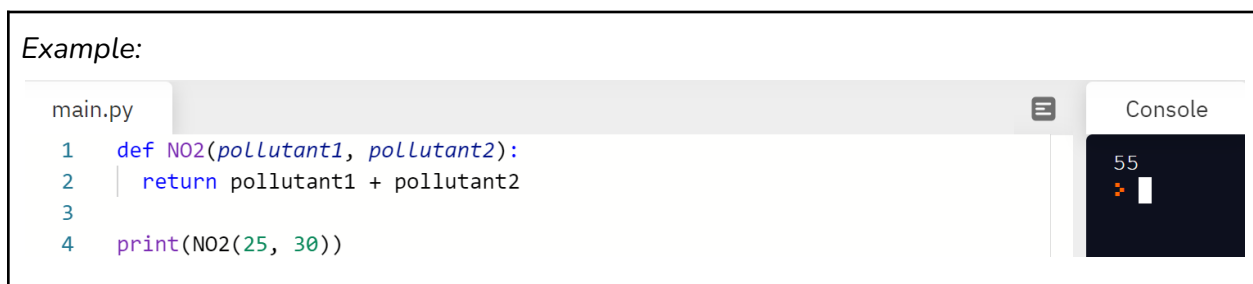
2. Indent the following line, and create a **return** statement that indicates that **adding** the two arguments is returned by this function.

```
    return pollutant1 + pollutant2
```

3. Print the function `NO2` and place the values 25 and 30 in the parentheses following the function `NO2` to print the total ppb of the two pollutants.

```
print (NO2(25, 30))
```

Example:

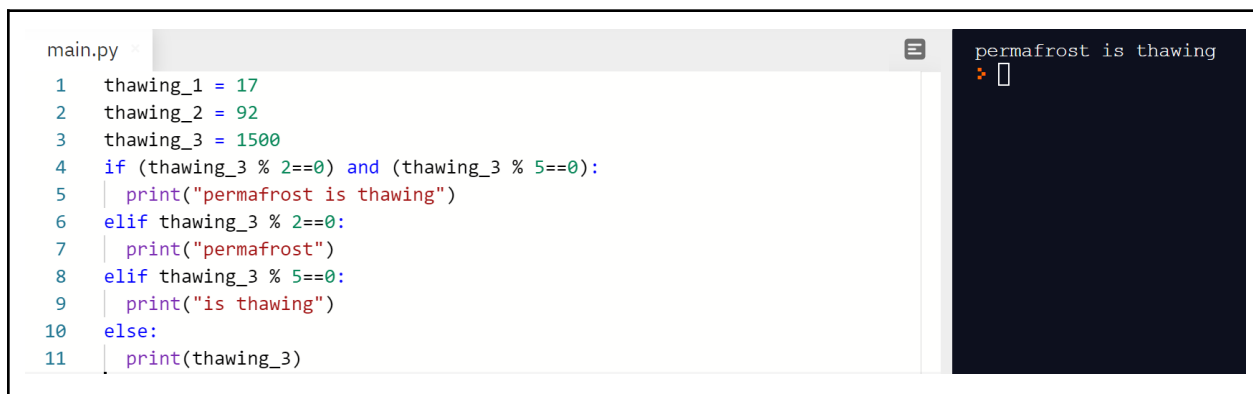


```
main.py
1 def NO2(pollutant1, pollutant2):
2     return pollutant1 + pollutant2
3
4 print(NO2(25, 30))
```

Console

```
55
```

Part 3: Transform logic into a function



```
main.py
1 thawing_1 = 17
2 thawing_2 = 92
3 thawing_3 = 1500
4 if (thawing_3 % 2==0) and (thawing_3 % 5==0):
5     print("permafrost is thawing")
6 elif thawing_3 % 2==0:
7     print("permafrost")
8 elif thawing_3 % 5==0:
9     print("is thawing")
10 else:
11     print(thawing_3)
```

permafrost is thawing

✓ **Task 1:** Using the permafrost example in lesson 5, transform this logic into a function and use the variable `years_thawing`.

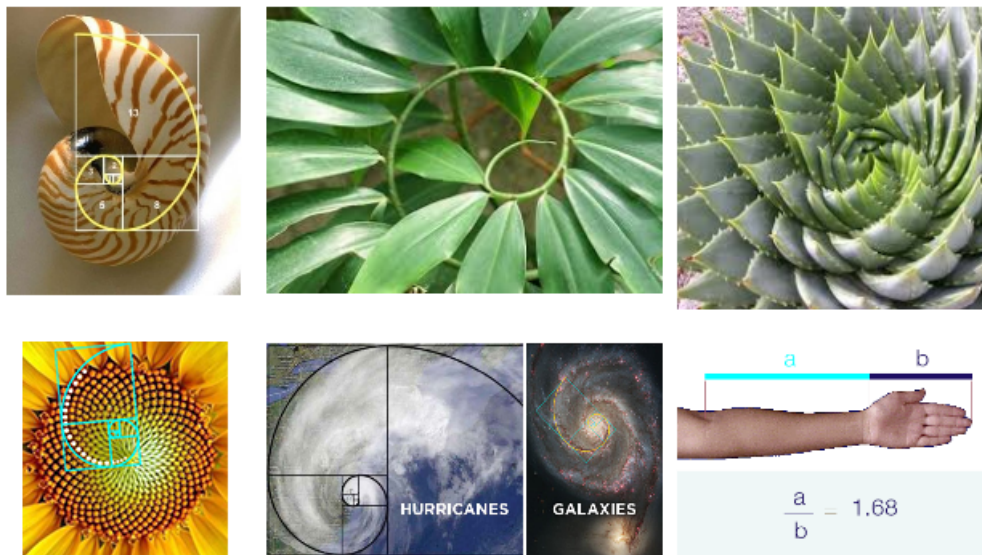
**Don't forget to indent if, elif, and else statements. Instead of using print, use return. After defining the function, print the result for the values 17 and 92 only.*

✓ **Task 2:** Print a boolean expression to assess whether `thawing_1` is equal to `thawing_2`.

Appreciating Nature - the Fibonacci numbers

The universe, though seemingly chaotic, can be governed by laws that create order. Through an understanding of the laws of the universe, one can build a deep appreciation for the patterns and order found in nature. The fibonacci sequence is a good example of this intersection of mathematics and nature. Starting with 0 and 1, this sequence continues by adding the last 2 numbers in the sequence: 1, 2, 3, 5, 8, 13, 21... This sequence alone has many interesting relationships, which are explored in detail in this [TED talk](#). For example, the ratio between the numbers approaches 1.618, otherwise known as [the golden ratio](#), the higher the fibonacci numbers become. For example, $2 \div 1 = 2$, $3 \div 2 = 1.5$, $5 \div 3 = 1.666$, etc.

Applications of the fibonacci numbers can be commonly found in nature. For example, the number of petals on different species of flowers follows the fibonacci sequence: lilies have 3 petals, rose hips have 5 petals, cosmea have 8 petals, daisies have 13 petals, chicory have 21 petals, and asteraceae have 34, 55 or 89 petals. The physical manifestation of this golden ratio, called the golden spiral, can be seen all over nature, from the shape of seashells to the galaxy.



Sources:

Carlson, S. C. (2019, November 14). Golden ratio. Retrieved December 16, 2020, from <https://www.britannica.com/science/golden-ratio>

Cubet. (2020, February 11). Golden Ratio In UI Design. Retrieved December 16, 2020, from <https://cubettech.com/resources/blog/golden-ratio-in-ui-design/>

Part 4: Use a Recursive Function

Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21...

The fibonacci numbers can be represented by the following mathematical equation:

$$F_n = F_{n-1} + F_{n-2}$$

As n in this equation is a subscript, it refers to the position of the number rather than the number itself. For example, the number 8 is the sum of the values in the two previous positions in the fibonacci sequence - the values of 5 ($n-1$) plus 3 ($n-2$). This is different from adding $n - 1$ (the value of 7) plus $n - 2$ (the value of 6) - this would be an entirely different equation.

$$F_n = F_{n-1} + F_{n-2}$$

$$F_8 = F_5 + F_3$$

Part 4: Create a function for the fibonacci sequence

1. Define the function fibonacci as a number.

```
def fibonacci (number):
```

2. To start the sequence with a default 0 and 1, create an `if` and `elif` statement to `return` 0 and 1.

```
    if number == 0:
```

```
        return(0)
```

```
    elif number == 1:
```

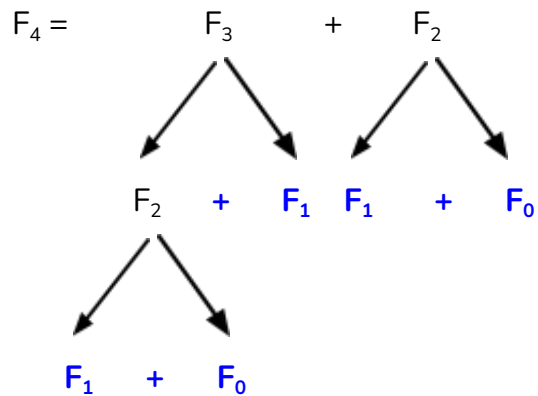
```
        return(1)
```

3. Create an `else` statement using the mathematical expression.

```
    else:
```

```
        return fibonacci(number -1) + fibonacci(number -2)
```

As you can see, this is a recursive function, meaning that a function is calling itself throughout the function. For example, in order to calculate the fibonacci of the number 4, you would need to calculate the fibonacci of 3 and 2, which would essentially bring you back to calculating the fibonacci numbers of 0 and 1, which satisfy the `if` and `elif` statements.



This establishes that the number in the fourth position in the fibonacci sequence is 3.

Fibonacci sequence	0	1	1	2	3
Position	0	1	2	3	4

- To print the fibonacci number in the fourth position in the sequence, make 4 the argument of the function.

Example:

Try inputting this function into a [visualizer](#). This will show you the sequence of commands.

```

main.py
1 def fibonacci(number):
2     if number == 0:
3         return 0
4     elif number == 1:
5         return 1
6     else:
7         return fibonacci(number - 1) + fibonacci(number - 2)
8
9     print(fibonacci(4))
  
```

Console: 3

✓ Task 3: Print the first 8 numbers of the fibonacci sequence.