

INF 110 **Discovering Informatics**

Functions

Functions

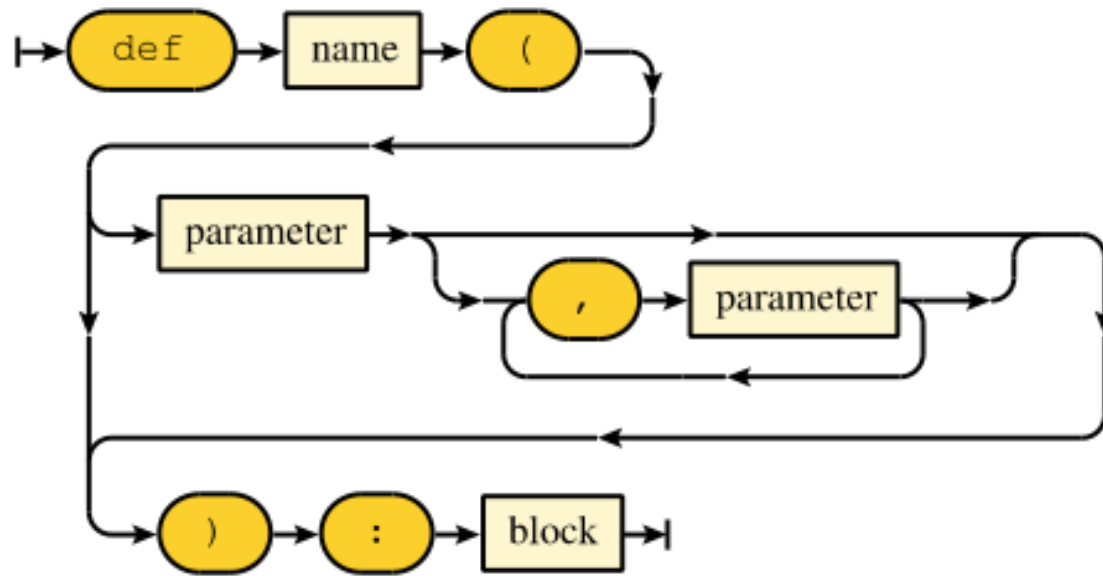
- A way of encapsulating logic and data into a reusable, callable piece of code
- For instance, `read_table()` is. Function we use a lot.
 - It reads comma separated values from a file and creates a table object out of them
- It provides the freedom to reuse the code without having to manually specify all the steps each time.

Functions

- The first step to using a function is to make sure that it exists.
- If we can't import an existing solution, we'll have to write a function.
- The process of creating a function is called “***function declaration***”.
- In the declaration stage, we are simply defining the steps that will be followed when someone executes the function.

Function Declaration

Syntax:



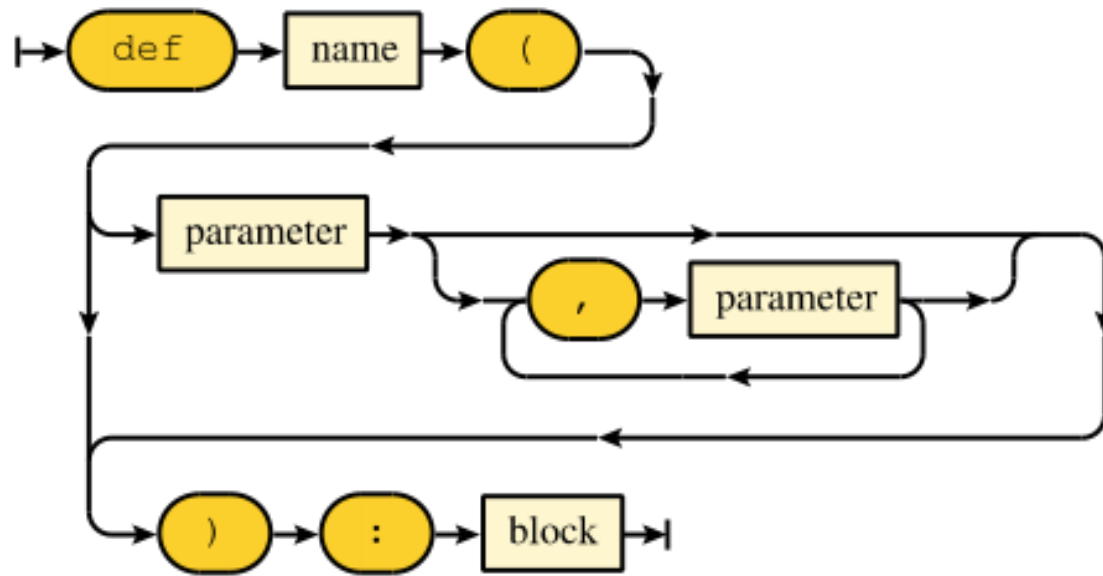
Example 1:

```
def is_chocolate(flavor):  
    return flavor == "chocolate"
```

- Step 1: specify the “def” keyword to let Python know we are defining a function.
- Step 2: give the function a (name should be all lowercase and no whitespace.)

Function Declaration

Syntax:



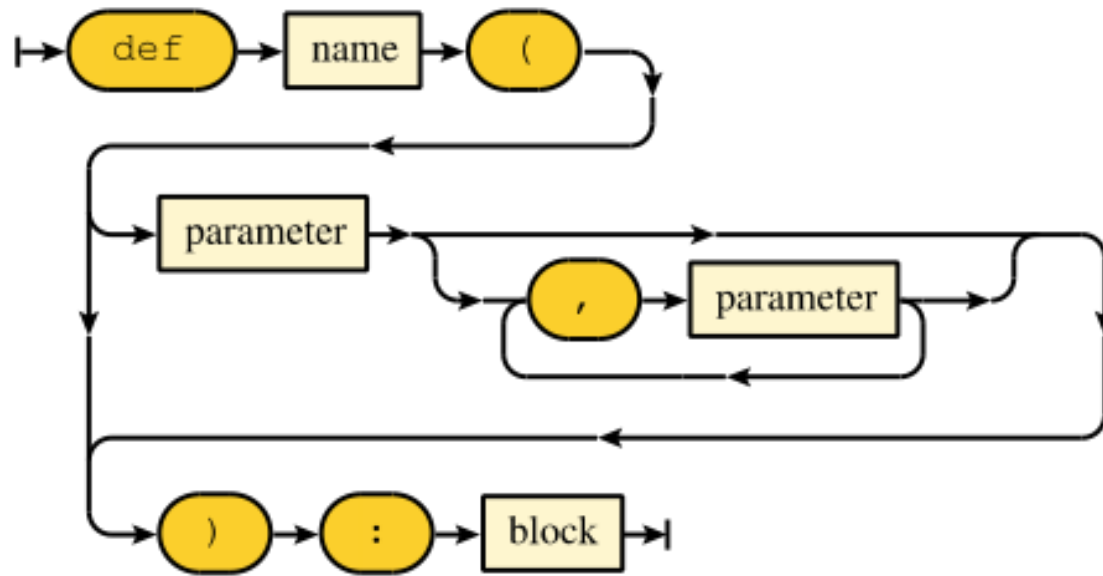
Example 1:

```
def is_chocolate(flavor):  
    return flavor == "chocolate"
```

- Step 3: open parentheses
- Step 4: specify parameters that the function will run on (in a comma separate list if there are multiple parameters)
- Step 4: close parentheses

Function Declaration

Syntax:



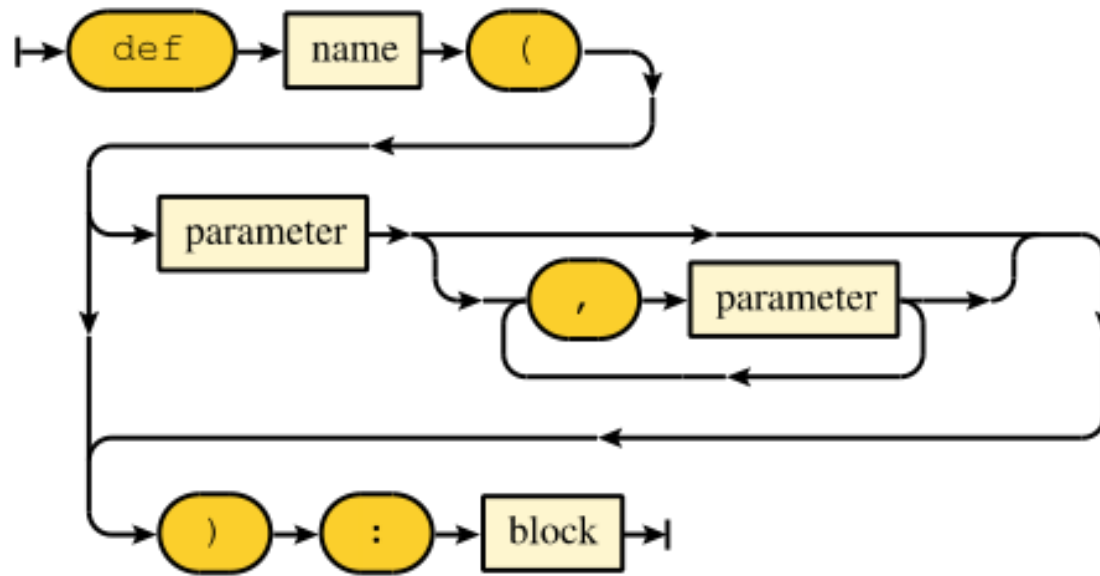
Example 1:

```
def is_chocolate(flavor):  
    return flavor == "chocolate"
```

- Step 5: add the function signature (which is a colon)
 - Function signature contains the function name, its parameters
- Step 6: Specify the body of the function (here it is a code block)

Function Declaration

Syntax:



Example 1:

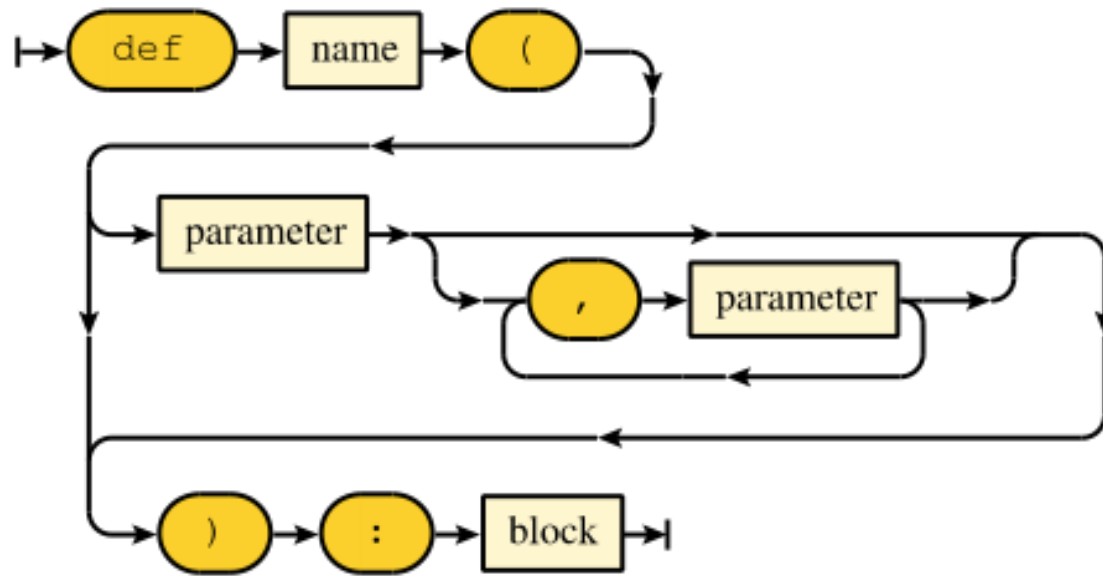
```
def is_chocolate(flavor):  
    return flavor == "chocolate"
```

Note: everything in the code block has to be indented since Python is parsed by whitespace

- Step 5: add the function signature (which is a colon)
 - Function signature contains the function name, its parameters
- Step 6: Specify the body of the function (here it is a code block)

Function Declaration

Syntax:



Example 1:

```
def is_chocolate(flavor):  
    return flavor == "chocolate"
```

This function checks whether the user has entered a word that is equal to “chocolate”.

In the function declaration, we have to name the parameter and then use it in the code block.

Functions are dumb and they don't know if you've defined this parameter before.

Live Code `is_chocolate?`

Tasks: Write the `is_chocolate()` function on the previous slide. Then apply it to different values and variables.

Learning Outcomes

- Writing simple functions
- Creating and using function parameters

Example 2: Spread

```
def spread(values):  
    return max(values) - min(values)
```

- Here, look at the parameters that are being passed in.
- The max and min of a single variable.
- So what does the "values" parameter have to be?

Example 3: Pig Latin

```
def pig_latin(word):  
    return word[1:] + word[0] + "ay"
```

- Here we are using a function to slice a list.
- Some “list” functions can be run on strings. This is because strings are a list of characters.
- We are using brackets to index the list from “1 colon”
 - “From the item at index 1 (which is the 2nd element), up to the end.

Example 3: Pig Latin

```
def pig_latin(word):  
    return word[1:] + word[0] + "ay"
```

Note: This is a good start but the pig latin rules are slightly more complicated. What words does this not work for?

Example 4: Quadratic Polynomial

```
def quadratic(x, a, b, c):  
    return a*x**2 + b*x + c
```

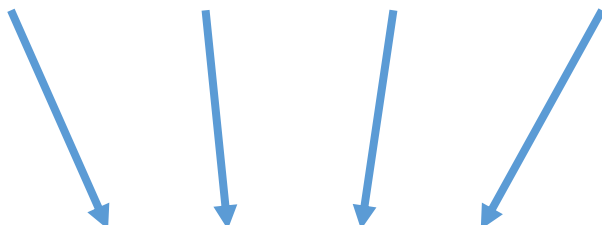
- This is an example of defining a function that require MULTIPLE PARAMETERS.
- Here, we are defining a quadratic function that requires 4 values.
 - x is the variable, and a,b,c are the coefficients

Positional Arguments

Means you call the arguments in the order listed in the parameter list:

```
quadratic(1.0, 2.0, 3.0, 4.0)
```

This is the default for Python.



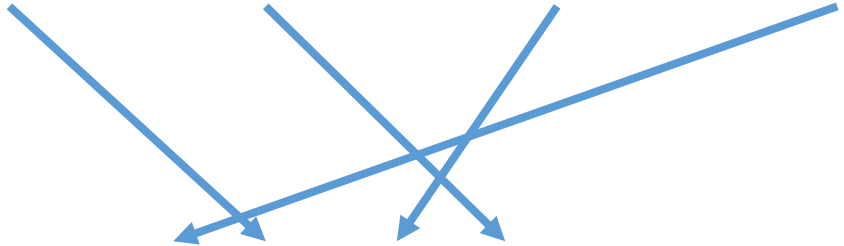
```
def quadratic(x, a, b, c):  
    return a*x**2 + b*x + c
```

Keyword Arguments

Means you call the arguments by specifying the parameter name:

```
quadratic(a=1.0, c=2.0, b=3.0, x=4.0)
```

```
def quadratic(x, a, b, c):  
    return a*x**2 + b*x + c
```

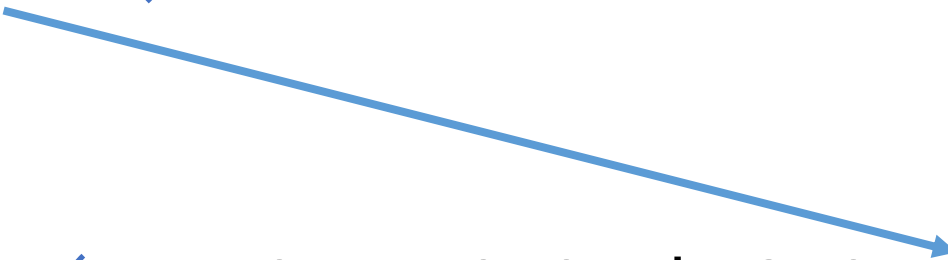


The diagram consists of four blue arrows pointing from the function call to the function definition. The first arrow points from 'a=1.0' to 'a'. The second arrow points from 'c=2.0' to 'c'. The third arrow points from 'b=3.0' to 'b'. The fourth arrow points from 'x=4.0' to 'x'.

Default Arguments

Means the function supplies one or more default values that can be absent from the call:

`quadratic(c=2.0)`



```
def quadratic(x=1.0, a=2.0, b=3.0, c=4.0):  
    return a*x**2 + b*x + c
```


Apply

The **apply** method creates an array by calling a function on every element in one or more input columns

- First argument: Function to apply
- Other arguments: The input column(s)

```
table_name.apply(my_function, 'column_label')
```

end