Functions

Defining a sequence of statements for code reuse

Alwin Tareen

What is a Function?

- A function is a named sequence of statements that performs a computation.

```
def displaygreetings():
    print("hello")
```

- When calling a function, you type the function's name, followed by parentheses, which can contain zero or more arguments.
- An argument is an input to the function.
- Then, the function performs some action, depending on its arguments.
- When there are muliple arguments to a function, you separate them with commas.

Built-in Functions

- There are a number of built-in functions that are included in `Python`, which are designed to solve common problems.
- These built-in functions include the following:
  - `print()`
  - `max()`
  - `min()`
  - `len()`
- You should consider the names of these built-in functions as reserved keywords. Do not use them as variable names.

The `max()` Function

The `max()` function will return the largest value in a collection of items.

- `max(42, 17)` $\rightarrow$ `42`
- `max(3.59, 8.24, 9.71, 6.53)` $\rightarrow$ `9.71`
- `max("abcdef")` $\rightarrow$ `"f"`

The `min()` Function

The `min()` function will return the smallest value in a collection of items.

- `min(26, 17, 57, 35)` $\rightarrow$ `18`
- `min(9.38, 4.75, 3.49, 8.75, 7.41)` $\rightarrow$ `3.49`
- `min("abcdef")` $\rightarrow$ `"a"`

The `len()` Function

The `len()` function returns the quantity of items in the collection. If the argument is a string, then `len()` returns the number of letters in the string.

- `len("pepperoni")` $\rightarrow$ `9`

Type Conversion Functions

- `Python` provides built-in functions that convert values from one data type to another.
- These built-in functions include the following:
  - `int()`
  - `float()`
  - `str()`

The `int()` Function

The `int()` function takes any value and converts it to an integer. If the given argument can't be converted, then `int()` exits with a `Traceback`(in other words, an error).

- `int("32")` $\rightarrow$ `32`
- `int("hello")` $\rightarrow$ `Traceback`

`int()` can convert floating-point values to integers. It retains the whole number part, and discards the fractional part.

- `int(3.9999)` $\rightarrow$ `3`
- `int(-2.3)` $\rightarrow$ `-2`

The `float()` Function

The `float()` function converts integers and strings to floating-point numbers.

- `float("32")` $\rightarrow$ `32.0`
- `float("3.14159")` $\rightarrow$ `3.14159`

The `str()` Function

The `str()` function converts integers and floating-point numbers to strings.

- `str(32)` $\rightarrow$ `"32"`
- `str(3.14159)` $\rightarrow$ `"3.14159"`

`str()` is useful when displaying numerical output to the user, because the concatenation operator only joins strings:

```
print("pi is: " + str(3.14))
```

Random Numbers

- The `random` module provides functions that generate pseudorandom numbers.
- To use the `random` module, you must include the following statement at the top of your program:

```
import random
```

A Note about Modules

- Python includes such a large number of functions, that they are organized into special groups called **modules**.
- Before using any functions from a module, you must import the module as demonstrated here:

```
import modulename
```

- To use a function from a module, you must type the module name, followed by a period, followed by the name of the function you want. For example:

```
num = random.randint(3, 8)
```

The `random()` Function

- `random()` returns a random floating-point number in the range: 0.0 to 0.99999999
- In mathematical notation, this is expressed as: $[0.0, 1.0)$

```
import random
num = random.random()
print(num)
```

The `randint(low, high)` Function

- `randint(low, high)` takes in two parameters: `low` and `high`.
- Then, it returns a random integer within the inclusive range of those two values.
- In mathematical notation, this is: [`low`, `high`]

```
import random
num = random.randint(1, 6)
print(num)
```

The choice() Function

- choice() selects an element at random from a collection of items.
- Usually, the collection is a **list data structure**. Lists will be covered in a later section.
- Note that in the following example, each of the numbers has a 25% chance of being selected.

```
import random
num = random.choice([18, 23, 9, 35])
print(num)
```

Math Functions

Python has a math module that provides most of the familiar mathematical functions that you would see on a calculator.

```
import math
```

Math Functions

The following are some of the mathematical functions that are included in the `math` module.

- `sqrt(x)`: This computes the square root of `x`.
- `exp(x)`: This computes the exponential function, e^x.
- `log(x)`: This computes the natural log function, $\ln x$
- `log10(x)`: This computes the logarithm base-10 function, $\log_{10} x$
- `sin(x)`, `cos(x)`, `tan(x)`: Computes the trigonometrical functions. The angle `x` must be expressed in radians, not degrees.
- Note that the mathematical constant π is included: `math.pi`

Creating Customized Functions

- The ability to define your own functions is a fundamental programming concept.
- Functions allow your program to become shorter, well organized, easier to read, easier to debug, and more reusable.
- A function definition specifies the name of a new function, and the sequence of statements that execute when the function is called.
- Once we define a function, we can reuse it over and over in our program.

The Structure of a Function Definition

- The first line of a function definition is called the **header**, and the rest is called the **body**.
- The header consists of the keyword `def`, then the function name, then parentheses, which may contain zero or more **parameters**.
- If there are multiple parameters in a function definition, then they must be separated by commas.
- The rules for naming functions are the same as the rules for naming variables(no punctuation, don't start with a number, etc.).
- There must be a colon at the end of the header.

The Structure of a Function Definition

- The following is an example of a function which has no parameters, and does not return a value.

```
def displaygreeting():
    print("hello world")
```

- Once you have defined a function, you can call it from anywhere in your program. You can even call a function from within another function.

```
displaygreeting()
```

- Note that the function definition must come before the function call in your program.

Functions which Return a Value

- Some functions perform actions and yield results. These require the use of the keyword `return`.
- When a function reaches the following line in its body:

```
return <value>
```

- The function stops executing, and returns `<value>` as its output.
- The body of a function may contain several `return` statements, but only the first one executed causes the function to exit.

```
def calculatesquare(side):
    return side**2
```

Flow of Execution

- The `Python` interpreter begins execution at the first statement of the program.
- Statements are executed one line at a time, in order, from top to bottom.
- Now that we have function definitions in our program, we have to keep in mind that the statements inside a function are not executed until the function is called.

Flow of Execution

- In a program with user-defined functions, execution will begin at the first statement which is outside any function.
- If a function call is made, the flow jumps to the body of the function, executes all statements there, and then comes back to pick up where it left off.
- Therefore, when reading a program, execution does not always proceed from top to bottom. Sometimes, it makes more sense to follow the flow of execution of the program.

Parameters and Arguments

- When a function is called, sometimes we must supply input values to that function, in between the parentheses. These are called **arguments**.
- However, when we define a function, we can have elements in between the parentheses called **parameters**.
- These parameters are actually variables which are local in scope to the function. This means that they cannot be used outside of that function.

Parameters and Arguments

```
def displaytwice(word):
    print(word)
    print(word)
```

- The parameter `word` can only be used inside the `displaytwice` function.
- The following is the function call:

```
displaytwice("hello")
```

- The argument `"hello"` is bound to the parameter variable `word`.
- Wherever `word` appears in the function, the value `"hello"` will be substituted.

Functions: End of Notes