Functions

CSC 1200 - Principles of Computing

Overview

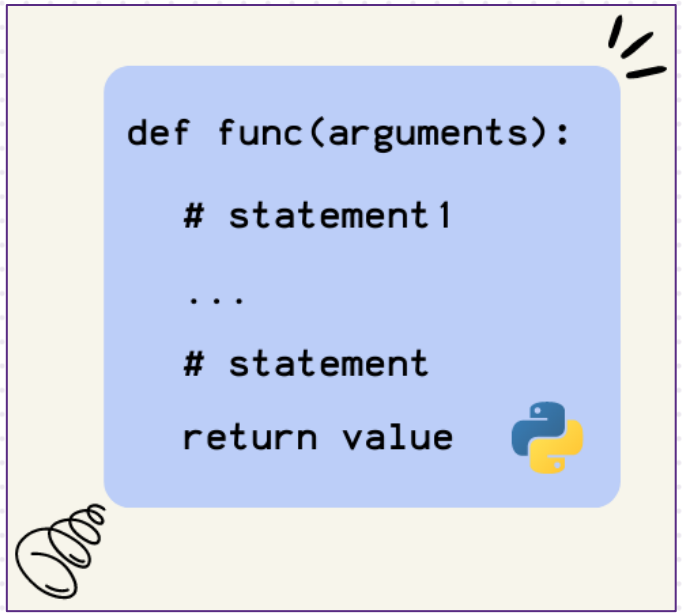
- Functions and Function Calls
- Type Conversion Functions
- Math Functions
- Function Composition
- Parameters and Arguments
- Return Values
- Stack Diagrams

Functions and Function Calls

- In programming, a **function** is a named sequence of statements that performs a computation.
  - When you **define** a function, you specify the name and the sequence of statements
  - Later, you **call** the function by name
  - Functions can have **arguments**, or values that are given to the function that are needed to perform the computation.
  - Functions can return a result, called the **return value**.

Here are some example built in Python functions:

- type(7)
- type('hi there')
- type(3.14)
- math.sin(90)
- math.cos(math.pi)
- math.sqrt(25)
- print('Hi there')
- print('The answer is', answer)
- print()



Type Conversion Functions

Python provides built-in functions that convert values from one type to another:

- int: takes a numeric or string value and converts it into an integer if it can
  - int('24')
  - int('two')
  - int(3.14)
  - int(2.71828)
  - int(-3.99)
- float: takes a value and converts it into a floating-point number if it can
  - float(24)
  - float('2.71828')
- str: converts its argument into a string
  - str(24)
  - str(2.71828)

Math Functions

A **module** (library) is a file that contains a collection of related functions.

Python has a math module that provides most of the familiar mathematical functions.

- To get access to the module, you must first import it using the statement import math.
- The above statement creates a module object named math.
- To access the functions and variables defined in the module, you must use **dot notation**.
  - math.sin is used to access the function for sine
  - math.pi is used to access the variable that represents the constant for pi
  - math.log
  - math.log10
  - math.e

Math Functions (Continued)

Here is a partial list of the functions in the math module:

math.**ceil**(*x*)	math.**perm**(*n*, *k=None*)	math.**cos**(*x*)
math.**comb**(*n*, *k*)	math.**exp**(*x*)	math.**asin**(*x*)
math.**fabs**(*x*)	math.**log**(*x*[, *base*])	math.**sin**(*x*)
math.**factorial**(*x*)	math.**log10**(*x*)	math.**atan**(*x*)
math.**floor**(*x*)	math.**pow**(*x*, *y*)	math.**tan**(*x*)
math.**gcd**(**integers*)	math.**sqrt**(*x*)	math.**degrees**(*x*)
math.**lcm**(**integers*)	math.**acos**(*x*)	math.**radians**(*x*)

Function Composition

- Recall that an **expression** is anything that has a value.
- If a function returns a value, then a function call is an expression and can be used anywhere that an expression is used.
- In mathematics, you have used function composition f(g(x)).
- You can use function composition in programming as well.
  - c = math.sqrt(math.pow(a,2) + math.pow(b,2))
  - print(sin(3*math.pi/2))

Adding New Functions

- So far, we have only looked at functions that are built-in to Python or provided in Python modules.
- You can add your own functions too.
- A function definition specifies the name of a new function, the arguments it needs, and the sequence of statements that execute when the function is called.

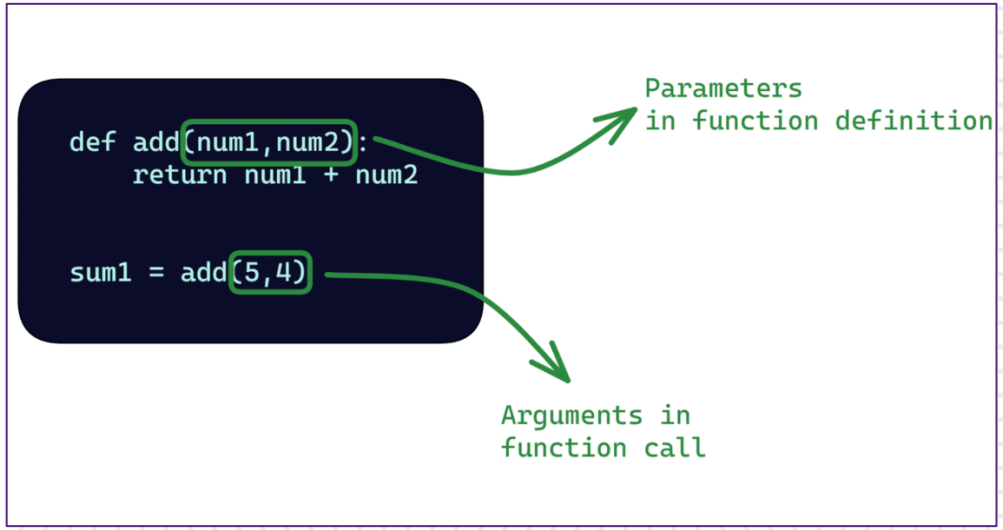
Format:

```
def function_name( parameter list ):    ← header
        statements ← body
```

Parameters and Arguments

- Some of the built-in functions we have seen require **arguments** (input values that get passed to the function when it is called).
  - print('Welcome')
  - math.sin(math.pi/2)
  - math.comb(total, 4)

- Inside the function, the values of the arguments are assigned to variables called **parameters** (variables the function uses to perform its computations).
  - Parameters are local to the function, meaning that they only exist inside the function
  - When variables are used as arguments they DO NOT have to have the same names as the parameters.
  - Expressions (anything with a value) can be used as arguments



Return Values

- The textbook distinguishes between two kinds of functions:
- Void functions - perform statements but don't produce a value (e.g., print('hi'))
- "Fruitful" functions - perform statements and yield a result (e.g., math.sin(0))
- Most of the time, functions yield results. In order to produce a result, the last line of the function should be:
    - return value

Example (try this out!):

```
def average( a, b ):
answer = (a + b) / 2
return answer
```

Stack Diagrams

- Long programs should be broken up into logical pieces with functions that implement each piece. This makes programs easy to write and easier to read/understand.
- It can be difficult to follow the flow of programs with lots of function calls (especially when functions call function that call functions that call …)
- To keep track of which variables are currently in use and what the values are, we use a **stack diagram**.
  - Show the value of each variable
  - Show the function each variable belongs to
  - Each function is represented by a **frame** (a box with the name of the function along with its variables and parameters)
- If an error occurs during a function call, Python prints the name of the function and the list of function calls made to get to the function. This is called a **traceback**.

Example

- cat is local to cat_twice

- If we try to access cat from print_twice, we will get a runtime error.

```
def cat_twice( part1, part2 ):
    cat = part1 + part2
    print_twice( cat )

def print_twice( bruce ):
    #print( cat )
    print( bruce )
    print( bruce )

##############################
# Main Program
# <module>
##############################

line1 = 'Bing tiddle '
line2 = 'tiddle bang. '
cat_twice( line1, line2 )
```

```
Bing tiddle tiddle bang.
Bing tiddle tiddle bang.
```

```
Traceback (most recent call last):
  File "C:/Users/tlee/AppData/Local/Programs/Python/Python39/Ch3
Stack Diagram Example.py", line 17, in <module>
    cat_twice( line1, line2 )
  File "C:/Users/tlee/AppData/Local/Programs/Python/Python39/Ch3
Stack Diagram Example.py", line 3, in cat_twice
    print_twice( cat )
  File "C:/Users/tlee/AppData/Local/Programs/Python/Python39/Ch3
Stack Diagram Example.py", line 6, in print_twice
    print( cat )
NameError: name 'cat' is not defined
```