

# Functions: Create

---

**CPSC 217: Introduction to Computer Science for Multidisciplinary  
Studies I**  
**July 2024**

Jonathan Hudson, Ph.D.  
Instructor  
Department of Computer Science  
University of Calgary

*June 4, 2024*

*Copyright © 2024*



# Functions

---

- A function is a block of code that accomplishes a single purpose, such as calculating an average from a list of values, or printing a message to console.
  - A function (optionally) takes some inputs (also named arguments), performs some operations, and (optionally) gives back a result.
- Proper design and use of functions can save time and make a program easy to read, maintain, organized, and efficient.



# Great job!

---

You've already been using functions

# You've already been using functions

Built-in functions are provided by the language

Library functions are similar but provided by another programmer

User-defined functions are created by programmers

All function operate the same

# You've already been using functions



All functions operate the same



Functions might receive some arguments to work with. This is handled as *parameters* in the function definition



When calling the function, we supply the actual data to the function as *arguments* in the function call



Functions may *return* results

# Functions Overview

---

Functions are useful because they:

1. Facilitate code reuse
  - Write once, use many times
2. Reduce code complexity
  - Allow programmers to break problems into smaller sub problems
  - Details relevant to solving a specific sub problem are placed in the function
  - Programmer can concentrate on higher level problems
3. Ease Maintenance
  - Bugs only need to be corrected once
  - Functions can be tested separately

# Python built-in functions

		Built-in Functions		
<code>abs()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>all()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>any()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>ascii()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bin()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>bool()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	
<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>	

# Good style function

---

- Functions are like tools:
  - They need to have a self descriptive name
    - **indicating a clear description of the task**
  - One function serves one purpose, code with different purposes should not be combined into one function.
    - **You do not design a fridge that is also a stove!**
  - Functions can use (call) each other
  - There could be multiple correct solution
    - It depends on your design



# Your functions

---

# User-Defined Functions

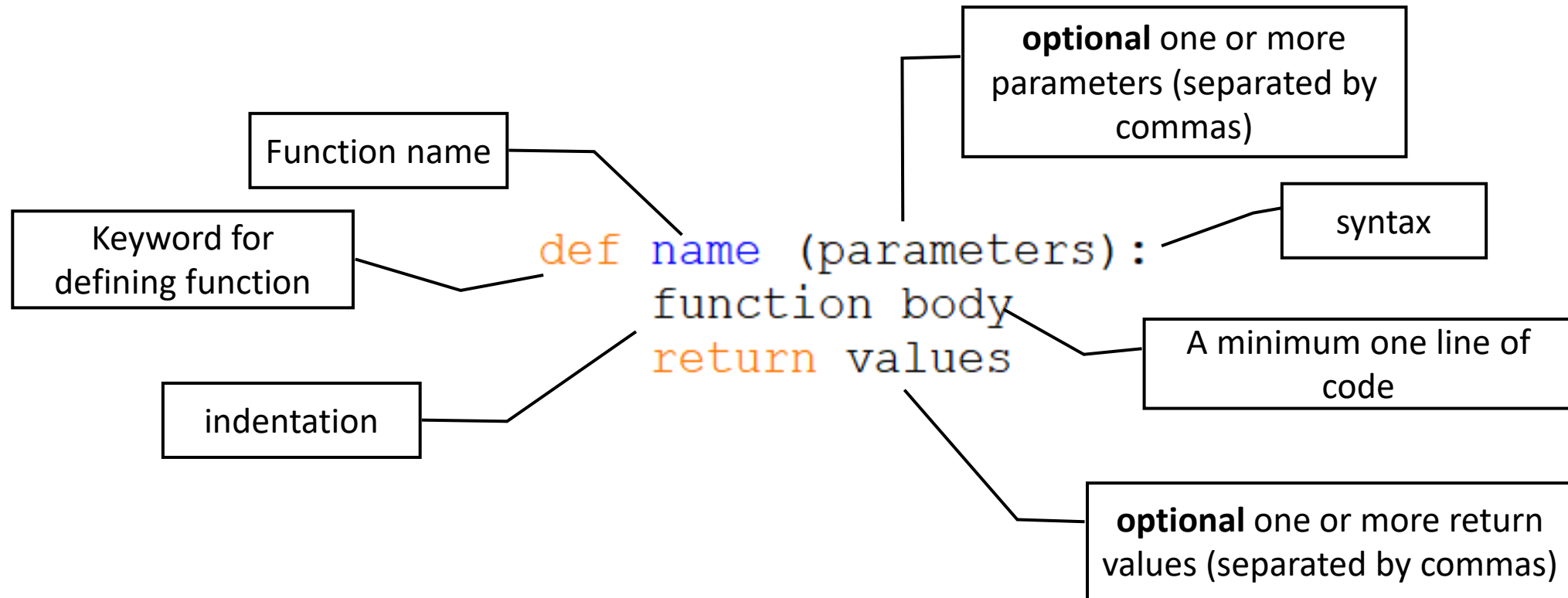
---

- Programmers can define their own functions.
- A function consists of the following components:
  - (Required) **Function name**
  - (Optional) **Function parameters**
  - (Required) **Function body** (code), including a return statement: The function's body must contain at least one line of code.
- The following is a minimal function that does nothing:

```
def foo():  
    pass
```

# User-Defined Functions

- You can define your own function using the following syntax:



# Storing functions


---

Think of what “import math” did

# Accessing functions in other libraries

---

```
def fun():  
    print ('Hello World!')
```




fun.py

```
def foo():  
    print ('Goodbye World!')
```



foo.py

```
from fun import *  
import foo  
def main():  
    fun ()  
    foo.foo()  
  
# Main body  
main()
```



main.py

# Design

---

# User-Defined Functions - Well Designed Functions

---

- A Function should:
  - Have a **descriptive name** that indicates the tasks the function performs.
  - **Serve one purpose**; code with different purposes should not be combined into one function.
  - **Reduce code redundancy**
    - A function that has one line of code is generally not a good function
    - A function is that used only once in code is generally not a good function
  - Start with **comments** that describe the function's purpose, the parameter(s), and any value(s) that will be returned.
- Functions can call other functions, which further helps in minimizing code redundancy.

# Parameters

---



# Parameters

---

- A parameter acts like a variable and holds the value we give to the function when we call it.

A parameter provides the needed data passed by the caller

```
def pay (amount):  
    print ("Direct bank deposit: $%d." % (amount))
```

```
def payroll ():  
    salary = 40 * 15  
    pay (salary)
```

```
payroll ()
```

# Parameters

---

- A parameter acts like a variable and holds the value we give to the function when we call it.

```
def pay (amount):  
    print ("Direct bank deposit: $%d." % (amount))
```

A parameter provides the needed data passed by the caller

```
def payroll ():  
    salary = 40 * 15  
    pay (salary)
```

**The parameter name and argument name can be different.**

The variable **salary** is a pointer to a place in memory

The parameter **amount** is a new variable created each time pay is called which is pointed to same information in memory

```
payroll ()
```

# User-Defined Functions - Multiple Parameters

---

- You can define a function that accepts multiple parameters
- The function call must match the number of parameters
  - It must also match the expected data type (not enforced by python)

```
def printbar(char, num):  
    bar = ''  
    for i in range(1, num + 1, 1)  
        bar = bar + char  
    print(bar)
```

```
printbar('-', 3)  
length = 10  
printbar('=', length)
```

Do these work?

- **printbar ('-')**
- **printbar (3)**
- **printbar (length)**
- **printbar (3, '-')**

# User-Defined Functions - Multiple Parameters

---

- The following function takes two points (x1, y1) and (x2, y2) in Cartesian plane and return the Euclidean distance between them.

```
def CalcDistance(x1, y1, x2, y2):  
    dx = x1-x2  
    dy = y1-y2  
    dSquared = (dx**2) + (dy**2)  
    result = dSquared ** (1/2)  
    return result  
  
print(CalcDistance(1, 2, 4, 6))
```

# User-Defined Functions - Multiple Parameters

```
#CONSTANTS
WIDTH = 800
HEIGHT = 600

def drawStar(pointer, length):
    for i in range(5):
        pointer.forward(length)
        pointer.right(144)

#Setup turtle
pointer = turtle.Turtle()
screen = turtle.getscreen()
screen.setup(WIDTH, HEIGHT, 0, 0)
screen.setworldcoordinates(0, 0, WIDTH, HEIGHT)
pointer.hideturtle()
screen.delay(delay=0)

pointer.up()
pointer.goto(400,300)
pointer.down()
drawStar(pointer, 100)
pointer.up()

screen.exitonclick()
```

# main() function

---

```
#CONSTANTS
WIDTH = 800
HEIGHT = 600

def drawStar(pointer, length):
    for i in range(5):
        pointer.forward(length)
        pointer.right(144)

def main():
    #Setup turtle
    pointer = turtle.Turtle()
    screen = turtle.getscreen()
    screen.setup(WIDTH, HEIGHT, 0, 0)
    screen.setworldcoordinates(0, 0, WIDTH, HEIGHT)
    pointer.hideturtle()
    screen.delay(delay=0)

    pointer.up()
    pointer.goto(400,300)
    pointer.down()
    drawStar(pointer, 100)
    pointer.up()

    screen.exitonclick()
```

main()

# Optional Parameters

---

# User-Defined Functions - Optional Parameters

---

- Optional parameters are parameters that programmers do not have to pass to the function during function call.
- They have default values that will be used if none were provided.
- Optional parameters should appear at the end of the parameter list in the function definition:

```
def printbar(char, num = 10):  
    bar = ''  
    for i in range(num + 1):  
        bar = bar + char  
    print(bar)
```

```
printbar('-')  
printbar('=', 20)
```



# User-Defined Functions - Optional Parameters

---

- You can define default parameter values
  - The function will always use these values unless new ones are declared

```
def foo(x=1, y=2):  
    print("x=", x, "y=", y)
```

```
foo()  
foo(3)  
foo(3, 4)
```

# User-Defined Functions - Optional Parameters

---

- You can define default parameter values
  - The function will always use these values unless new ones are declared

```
def foo(x=1, y=2):  
    print("x=", x, "y=", y)
```

```
foo()           x= 1 y= 2  
foo(3)         x= 3 y= 2  
foo(3, 4)      x= 3 y= 4
```

# User-Defined Functions - Variadic Functions

---

- Variadic functions are functions that accept a variable number of parameters.
- Useful when you do not know *a priori* how many parameters the user may pass to the function:
- Commonly used for functions that perform some operation on a series of inputs, such as summing numbers, concatenating strings, or formatting output.
  - Ex. `print("My name is", name, ". I am from", country)`

# User-Defined Functions - Variadic Functions

---

- You can define a variadic function in python as follows:

```
def variadicFunction(*args):  
    for i in args:  
        print(i)
```

- *args* is a conventional name. You can use any name you like.
- You can pass any number of arguments when calling the function:

```
variadicFunction("a")           →    a  
variadicFunction("a", "b", "c") →    a  
                                b  
                                c
```

# User-Defined Functions - Variadic Functions

---

- You can include defined parameters at the beginning as follows:

```
def variadicFunction(param1, *args) :  
    print("Named param is:", param1)  
    for i in args:  
        print(i)
```

- Start with any named parameters you may have followed by the variable-length parameter list. You can have none.
- You must pass the named params followed by any number of arguments when calling the function:

```
variadicFunction("a") → Named param is: a
```

```
variadicFunction("a", "b") → Named param is: a  
b
```

# Onward to ... using functions.

---

Jonathan Hudson  
[jwhudson@ucalgary.ca](mailto:jwhudson@ucalgary.ca)  
<https://cspages.ucalgary.ca/~jwhudson/>



UNIVERSITY OF  
CALGARY