CS65: Introduction to Computer Science

Documenting Functions
Scope: Local Variables and Global Variables
Keyword Arguments
Default Parameters

Md Alimoor Reza
Assistant Professor of Computer Science

Last Lecture

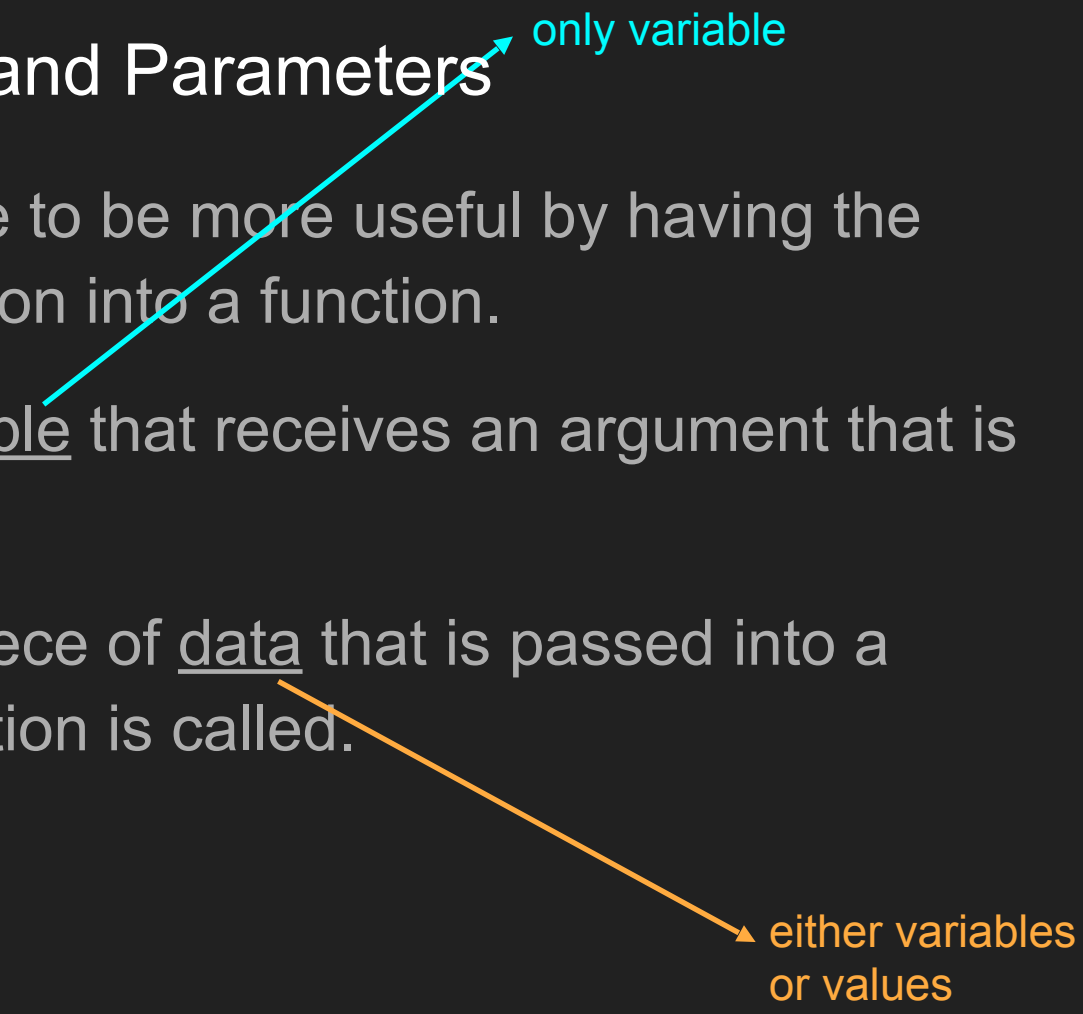
- Transfer of Control
- Functions Parameters and Arguments
- Functions Multiple Arguments
- Value-Returning Functions

Review: Arguments and Parameters

Functions can be made to be more useful by having the ability to pass information into a function.

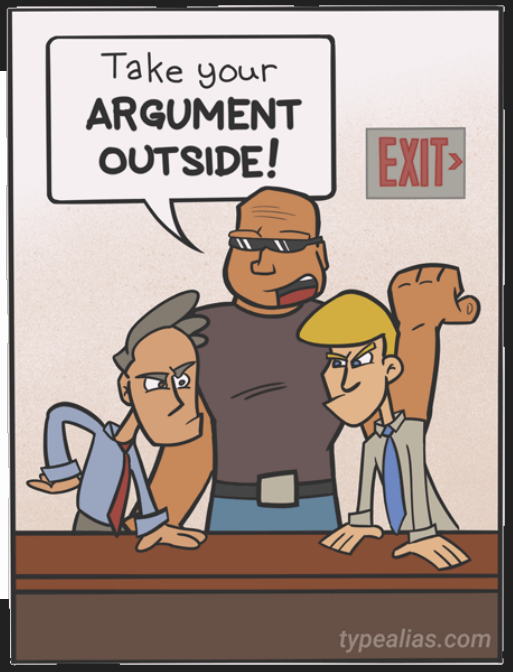
A **parameter** is a variable that receives an argument that is passed into a function.

An **argument** is any piece of data that is passed into a function when the function is called.



parameter

```
4  def vote_verification(age):
5      if age >= 18:
6          print("You can vote!")
7      else:
8          print("You are not old enough to vote")
9
10 # ask user for age and call the function
11 val = int(input("How old are you? "))
12 vote_verification(val)
13
```



argument

Review: Multiple Arguments

Often it's useful to write functions that can accept multiple arguments.

When multiple arguments or parameters are used, the values are separated by a comma in the listing between the parentheses ().

The order is important: the first argument will correspond to the first parameter, the second argument will correspond to the second parameter, and so on.

Functions5.py

```
def show_exponent(num1, num2):
    result = num1**num2
    print(result)

def main():
    print("2 raised to the 5th power is")
    show_exponent(2, 5)

main()
```

Review: Exercises

#1

- Define a function called greeting()
- The function should first prompt the user for their name
- The function should then print “hello,” followed by the name that was entered.

#2

- Create a function called `coin_flip()`
- Simulate a flip of a coin
- 50% of the time it should print “Heads”
- 50% of the time it should print “Tails”
- Use a loop to call coin_flip() 10 times

```
import random

def coin_flip():
    number = random.randint(1,2)
    if number == 1:
```

#3

- Create a function called roll_die
- use a parameter that indicates the number of sides on the die
- Simulate the rolling of a 6-sided die and 20-sided die

```
>>> %Run Challenge4.py

  6-sided die...
you rolled a 2
20-sided die...
you rolled a 9
```

Review: Syntax for returning values

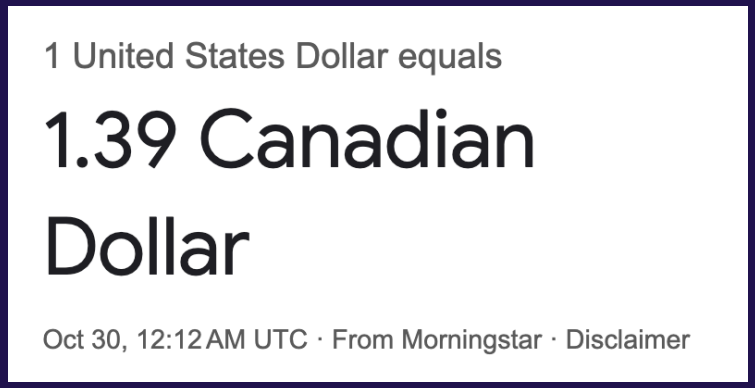
To return a value from a function you've defined, put the **`return`** statement at the end of your function (followed by a value or an expression)

- Whenever the return statement is executed, the corresponding value is returned to the code that called the function

```
2  # intro to value-returning functions
3
4  def weight_on_moon(earth_weight):
5      result = earth_weight * 0.165314
6      return result
7
8  moon_weight = weight_on_moon(180)
9  print("On the moon, I would weigh", moon_weight,"pounds!")
0
```

ReturnFunctions1.py

Review: Exercise #4



Define a function called **`convert_to_canadian`**

The function should have a *parameter* called us_dollars

The function should then return the parameter * 1.43

Prompt the user for an amount of money and output the conversion.

```
>>> %Run Challenge1.py
 How much to convert? 100
 100 US dollars converts to 143.0 Canadian dollars
```

Why should you return instead of just printing the result?

Returning values gives you more flexibility in how the function can be used.

The function can be seen as a “black box”

Given some input parameters, returns a value (that you can use however you want)

Today's Plan

- Documenting Functions
- Scope: local vs. global variables
- Keyword Arguments
- Default Parameters

Documenting Functions

It is *always* a good idea to comment what a function does, what parameters are, and what it will return

To do it, add a **`docstring`**, a multiline comment (“””) starting as the first line in your function.

Things typically included:

- a description of the function's purpose
- a description of each parameter - so that they know what to pass as arguments
- a description of the return value, if any - so they know how to use what they get back

Review: Exercise #5

Write a function named `c_to_f` that takes a parameter named celsius and returns the Celsius temperature to Fahrenheit temperature. The formula is a follows:

$$F = \frac{9}{5}C + 32$$

Convert 100 degrees Celsius to Fahrenheit

```
>>> %Run Challenge2.py
 100 degrees Celsius is 212.0 Fahrenheit
```

Use a loop and the function to output:

```
0 --- 32.0
10 --- 50.0
20 --- 68.0
30 --- 86.0
40 --- 104.0
50 --- 122.0
60 --- 140.0
70 --- 158.0
80 --- 176.0
90 --- 194.0
100 --- 212.0
```

Documenting Functions

ReturnFunctions2.py

```
2  # example of using comments (docstrings) in a function
3
4  def f_to_c(fahrenheit_temp):
5      """
6      Convert a temperature from Fahrenheit to its Celsius equivalent.
7
8      Paremeters:
9          fahrenheit_temp: a float, the Fahrenheit temperature to be converted
10
11     Returns:
12         a float, the temperature converted into Celsius
13     """
14     celsius_temp = (fahrenheit_temp-32)*(5/9)
15     return celsius_temp
16
17 cel = f_to_c(212)
18 print("212 degrees Fahrenheit is", cel, "degrees Celsius")
```

Exercise #1

Write a function, called **`gallons_of_paint`**. It should have three parameters: width, length, and height (to be input in feet)

The function should calculate the number of gallons of paint needed to cover the walls of the room (but not the floor nor ceiling)

Hints:

- the total surface area of the walls of the room:

```
width*height*2 + length*height*2
```

- one gallon can of paint will cover 400 square feet

Test your code to make sure it's correct

Add appropriate comments (and a docstring) in your function

Today's Plan

- Documenting Functions
- Scope: local vs. global variables
- Keyword Arguments
- Default Parameters

What's wrong with this code?

```
3  def my_function(add_val):
4      my_var = 26
5      result = my_var + add_val
6      print(my_var, "plus", add_val, "is", result)
7
8  my_function(2)
9  print(my_var)
10
```

Scope1.py

Warning, Trick Question: what will be output here:

```
num_students = 350

def first_grade_classroom():
    num_students = 30
    print(num_students)

def second_grade_classroom():
    print(num_students)

first_grade_classroom()
second_grade_classroom()
```

Scope2.py

Scope

A variable's scope is the part of the program that the variable is visible to.

In Python, a scope can be local to a function or global (visible everywhere).

Thus, if you first declare a variable *inside* a function, it's scope is local, and it can ***only*** be accessed *inside* the function

Global Variables

If you really want to *change* a global variable inside a function, you have to use the **`global`** keyword.

Scope3.py

```
# experiments with global variables

num_students = 350 #this variable is global because it is defined outside of a function

def first_grade_classroom():
    global num_students #here, we tell it to really use the global variable
    num_students = 30
    print(num_students)

def second_grade_classroom():
    print(num_students)

first_grade_classroom()
second_grade_classroom()
```

```
>>> %Run Scope3.py
30
30
```

Local vs. Global Variables

In general, global variables are to be *avoided*

Keep all of your code (including variables) inside of functions – use local variables

Why?

- As programs grow in size, it's difficult to keep all of the global variables in mind
- With a global variable, any function could possibly change its value
    - the repercussions could cause other code depending on it's value to break
- It makes developing code in teams difficult
    - You would need to communicate all of the variable names with your team all the time

Today's Plan

- Documenting Functions
- Scope: local vs. global variables
- Keyword Arguments
- Default Parameters

Multiple Parameters Review

Recall that when using multiple parameters, order matters!

```
def split_bill(total, tax_percent, tip_percent, num_people):
    amount = (total + total*tax_percent + total*tip_percent)/num_people
    return amount

amount_per_person = split_bill(100, .06, .15, 4)
print("each person owes $", amount_per_person, sep="")
```

KeywordArguments1.py

Keyword Arguments

Keyword arguments allows the calling argument to be specified by the name of the parameter, rather than the position in the call to the function.

Keyword Arguments

Keyword arguments allows the calling argument to be specified by the name of the parameter, rather than the position in the call to the function.

```
def split_bill(total, tax_percent, tip_percent, num_people):
    amount = (total + total*tax_percent + total*tip_percent)/num_people
    return amount

# note by using keyword arguments, the order does not matter
amount_per_person = split_bill(tax_percent = .06, num_people = 4, total = 100, tip_percent = .15)
print("each person owes $", amount_per_person, sep="")
```

KeywordArguments2.py

Keyword Arguments

Keyword arguments allows the calling argument to be specified by the name of the parameter, rather than the position in the call to the function.

```
def split_bill(total, tax_percent, tip_percent, num_people):
    amount = (total + total*tax_percent + total*tip_percent)/num_people
    return amount

# note by using keyword arguments, the order does not matter
amount_per_person = split_bill(tax_percent = .06, num_people = 4, total = 100, tip_percent = .15)
print("each person owes $", amount_per_person, sep="")
```

Exercise #2 Keyword Arguments

A high school needs to repaint its gymnasium. Assume that a room has:

Length = 106

Width = 90

Height = 24

Use keyword arguments to call your gallons_of_paint function with the `height` argument in the first position, and the `width` in the second position, and and the `length` in the third position

Today's Plan

- Documenting Functions
- Scope: local vs. global variables
- Keyword Arguments
- Default Parameters

Default Parameters

Sometimes a function may have a parameter that is *almost always* called with the same argument.

Python allows the definition of a function to supply a default parameter values

If the parameter corresponding to an argument is not specified in the function call, the default value is used

The use of default parameters allows the use of the function to be very versatile. The number of arguments supplied can be variable

Default Parameters

```
1  # T. Urness
2  # default parameters
3
4  def split_bill(total, tax_percent = .06, tip_percent = .15, num_people = 2):
5      amount = (total + total*tax_percent + total*tip_percent)/num_people
6      return amount
7
8  # use the default parameters!
9  amount_per_person = split_bill(70)
10 print("each person owes $", amount_per_person, sep="")
```

DefaultParameters.py

Lab 7

For full credit, each of your functions should have complete docstrings

- multi-line comments at the top of each function that specifies the name of the function, a description of the parameters, and a description of the return value)

The file you submit in CodePost must be named exactly Lab8.py (note the capitalization) and it must contain functions named exactly **`letter_grade`** and **`letter_grade_extra_credit`** and **`convert_to_liters`** and **`bmi`**

```
A
B
input error
F
```

```
invalid
A
B
D
input error
```

```
75.7082
```

```
Overweight
```

Announcement

Lab #7 out today; due on Saturday, November 8th

Text Reading

https://python.swaroopch.com/functions.html