CS65: Introduction to Computer Science

Functions with Parameters
Value-Returning Functions

Md Alimoor Reza
Assistant Professor of Computer Science

Last Lecture

- String Methods
- Introduction to Functions

Useful string methods

- **Syntax:** string_expression.method_name($argm_1$, $argm_2$, $argm_n$)

method	purpose	returned value
s.upper() s.lower()	converts letters to upper or lower case	modified copy of s
s.startswith(svar[,start[,stop]]) s.endswith(svar[,start[,stop]])	is svar a prefix/suffix of s?	Boolean value
s.join(iterable)	concatenates strings from iterable, with copies of string s inbetween them	string result of all those concatenations/ interspersings
s.split(sep)	get list of strings obtained by splitting s into parts at each occurrence of sep	list of strings from between occurrences of sep
s.replace(old, new[,count])	replace all (or count) occurrences of old str with new str.	string with replacements performed

Review: Modules

One of the things that makes Python such a great language is that there are many functions you can use that aren't built in.

They're written by other programmers and made available to you.

In order to use them, you need an **`import`** statement.

Something you import this way is called a module.

A module is some pre-defined code that can be available to a program through the keyword **`import`**

```
4 from graphics import *
```

Review: Module Examples

```
import random
print( random.randint(1,100) )
```

In random.randint(1,100),

random is the name of the module and **randint** is the name of the *function* we want to use from that module.

1 and 100 are ***arguments*** we pass to the function that tell it what range of numbers we want our random number to be in.

```
from graphics import *
```

Notice the use of *

In this case, we don't need to prepend calls to the graphics library with `graphics.function_name()`

Review: Abstraction in Computer Science

In computer science, you build bigger, more complex things by using the building blocks others (or you) have provided.

Abstraction examples we've already seen:

We've already used several built-in ***functions*** in this class:

- `print("Hello")`
- `val = int(input("please enter a number")`
- `my_list = [2,3,4]`
- `num_elements = len(my_list)`
- `max_num = max(my_list)`

These are "built-in" functions in Python – they are part of the language and you don't need anything else to use them

Review: Calling Functions

- `print("Hello")`
- `val = int(input("please enter a number")`
- `my_list = [2,3,4]`
- `num_elements = len(my_list)`
- `max_num = max(my_list)`

Why are these awesome?

- We don't have to think about the code that makes these work in order to use them (abstraction)
- The code these programmers wrote is reusable
  - the same code they wrote once gets used in many many programs by many different programmers

Review: Defining your own functions

The syntax for defining your own function includes the following

- keyword **`def`** (short for define)
- a name/identifier that you pick (same rules as variable names)
- parentheses ( ) (later, we'll put things inside these parentheses)
- colon :
- *indented* block of code

Review: Demo Functions1.py

This function won't run or do anything until you call it. Call it just like any other function you've used.

```
# the following defines a function
def message():
    print("Hello CS 65!")
    print("I'm in a function!")

message() # call the function
```

Functions1.py

Review: Exercise#1

In Thonny, open a new file.

Create a function named **`my_function`**

When **`my_function`** is called, it should print out “THIS IS AWESOME!!”

Use a loop; call the function so it prints out “THIS IS AWESOME!!” 10 times

What is the minimum number of lines of code you can write this program?

Disclaimer

The best way to really understand this material is to practice it, and practice it some more

At first, it may feel confusing, unnatural

But trust that it gets easier with more examples you see and the more you practice it.

And the payoff is coming…

Today's Plan

- Transfer of Control

Transfer of Control

When a function gets called, the execution of the code is transferred to the statements within the function definition

After, the execution returns to the line below where the function was called

Transfer of Control

When a function gets called, the execution of the code is transferred to the statements within the function definition

After, the execution returns to the line below where the function was called

What will be output?

```
def message():
    print("Hello CS 65!")
    print("I'm in a function!")

print('here')
message() # call the message function
print("thank you for playing")
```

Functions2.py

Today's Plan

- Transfer of Control
- Functions Parameters and Arguments

Arguments

In most of the functions we've called, we've passed arguments into them. Arguments are data that we put inside of the parentheses that the function needs to work as intended.

Argument examples:

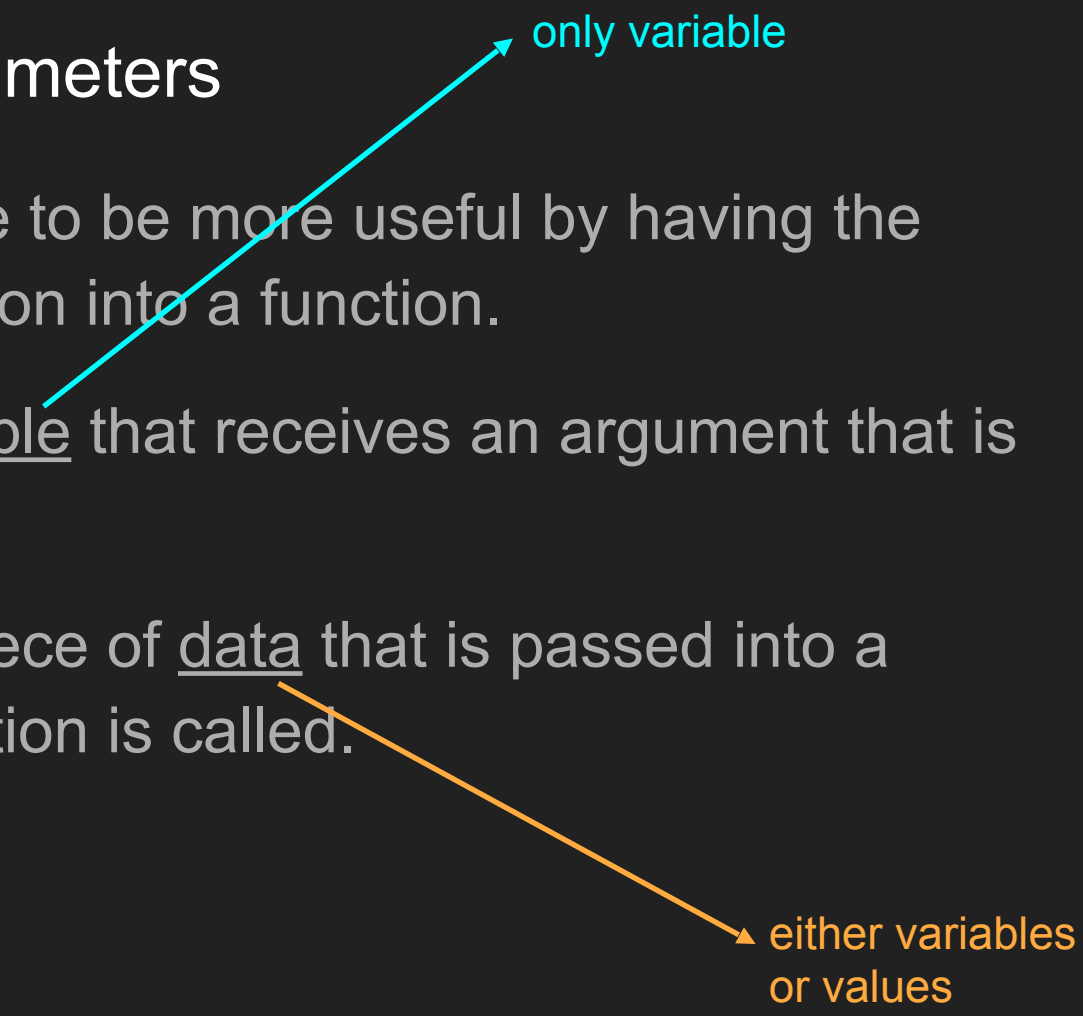
```
print("Hello world!")
num_cookies = int(input("How many cookies do you want to bake? "))
range(10)
name = input("What is your name")
len(name)
```

Arguments and Parameters

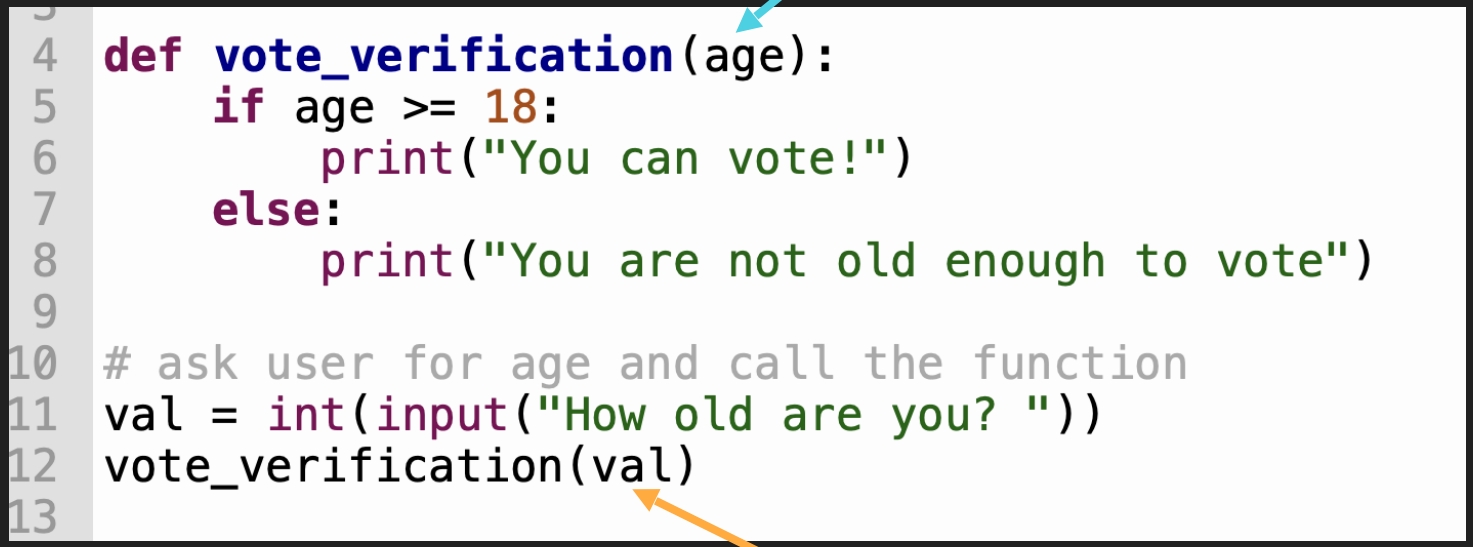
Functions can be made to be more useful by having the ability to pass information into a function.

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.

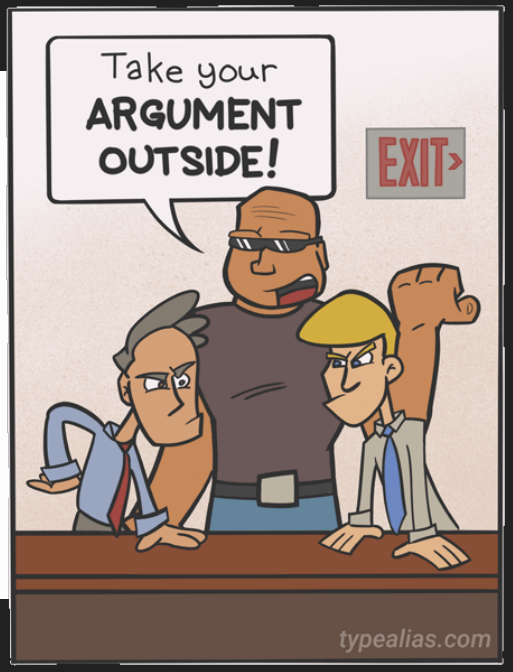


Functions4.py



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

Today's Plan

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

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()
```

Exercise#1

Define a function called **`greetings()`**

- The function should first prompt the user for their name
- The function should then print “hello,” followed by the name that was entered.

Call the function two times

```
1  # Alimoor Reza
2  # function with greetings
3
4
5
6
7
8  greetings()
9  greetings()
```

Example output:

```
>>> %Run exercise1.py

  Enter your name: Alimoor Reza
  Alimoor Reza
  Enter your name: Professor Reza
  Professor Reza
```

Exercise#2

Create a function called `flip_coin()`

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 `flip_coin()` 10 times

Hint:

```
1  import random
2
3  def flip_coin():
4      # your code for challenge #2
5      num = random.randint(1,2)
```

Output:

```
>>> %Run exercise2.py

  you flipped a Tail
  you flipped a Tail
  you flipped a Tail
  you flipped a Tail
  you flipped a Head
  you flipped a Head
  you flipped a Head
  you flipped a Tail
  you flipped a Head
  you flipped a Head
```

Exercise#3

Create a function called **`roll_die`**

- Function should use a parameter that indicates the number of sides on the die
- Print out a random number between 1 and the parameter (inclusive)

Simulate the rolling of a 6-sided die

Simulate the rolling of a 20-sided die

Your code should include…

```
print("6-sided die...")
roll_die(6)

print("20-sided die...")
roll_die(20)
```

Possible outputs:

```
>>> %Run Challenge4.py

  6-sided die...
  you rolled a 3
  20-sided die...
  you rolled a 11
```

```
>>> %Run Challenge4.py

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

#1

- Define a function called **`greetings()`**
- 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 **`flip_coin()`**
- 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

#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
```

In-Class Exercise

What number will be output?

```
1  import random
2
3  # sets the seed value for the random number generator
4  # so all future random numbers will have the same sequence
5  random.seed(0)
6
7  def roll_die(sides):
8      # your code for challenge #4 goes here
9      [illegible]
10     [illegible]
11
12 roll_die(1000)
```

Today's Plan

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

Return values: Getting data out of functions

Parameters & Arguments: Enable getting data *into* functions

Return values: Enable getting data out of functions - think of it as the *result* of a function

Return values: Getting data out of functions

Parameters & Arguments: Enable getting data *into* functions

Return values: Enable getting data out of functions - think of it as the *result* of a function

Let's think about examples from functions we've called:

```
name = input("What is your name? ")
name_length = len(name)
print(name,"has",name_length,"letters")
```

`input()` returns a string (whatever the user typed) and we saved that result to `name`

`len()` returns an integer (the length of the string name) and we saved that result to `name_length`

print() doesn't return anything - we never see print on the right side of an =

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

Another Example?

Let’s do this together

Create a function, named **`miles_to_empty`**

The function should take in one parameter, `gallons`, which corresponds to the number of gallons left in the tank

Assuming the car gets 27.5 miles per gallon, the function should return the number of miles you can drive until the gas tank would be empty.

If the input, `gallons`, is less than zero, or greater than 12, the return value should be -1.

Test out your function with various examples.

1 United States Dollar equals

1.39 Canadian Dollar

Oct 30, 12:12 AM UTC · From Morningstar · Disclaimer

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 * 1.39

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

```
>>> %Run exercise4.py

  How much to convert? 100
  100.0 US dollars converts to 139.0 Canadian dollars.
```

Exercise #4 hint

Here is the “driver” code … now you write the function

```
1  def convert_to_canadian(usd_param):
2
3      # your Python code here …
4
5  usd_argument = float(input("How much to convert? "))
6  result = convert_to_canadian(usd_argument)
7  print(f"{usd_argument} US dollars converts to {result} Canadian dollars.")
```

In-Class Exercise

Use your function to determine the exchange amount for 14 US dollars to Canadian dollars.

In other words, what will be returned by:

```
convert_to_canadian(14)
```

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:

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

Use a loop and the function to output:

```
>>> %Run exercise5.py

  0 degree Celcius is 32.0 Fahrenheit.
 10 degree Celcius is 50.0 Fahrenheit.
 20 degree Celcius is 68.0 Fahrenheit.
 30 degree Celcius is 86.0 Fahrenheit.
 40 degree Celcius is 104.0 Fahrenheit.
 50 degree Celcius is 122.0 Fahrenheit.
 60 degree Celcius is 140.0 Fahrenheit.
 70 degree Celcius is 158.0 Fahrenheit.
 80 degree Celcius is 176.0 Fahrenheit.
 90 degree Celcius is 194.0 Fahrenheit.
 100 degree Celcius is 212.0 Fahrenheit.
```

Why should you return instead of just printing the result?

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)

A motivating example

Consider the pay calculator.

```
def calculate_pay(wage,hours):
    if hours <= 40:
        pay = wage*hours
    else:
        overtime_hours = hours-40
        pay = (wage*40) + (wage*1.5*overtime_hours)
    return pay
```

We might want to allow a user to interact with it to check their pay; or, we might read employee information from a file so that we can process all their paychecks.

Time to work

Lab #6 due on Monday, November 3rd

The file you submit in Xuexitong must be named exactly Lab6.py (note the capitalization) and it must contain functions named exactly **`introductions`** and **`sleep_counter`** and **`print_num_vowels`**

Text Reading

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

For Loop Example

```
my_string = input("enter a string: ")

for ch in my_string:
    print(ch)
```

```
enter a string: asdf
a
s
d
f
```

Example: How many spaces in a string

```
# starter code and not-so-subtle hint
def print_spaces_count(input_string):
    count = 0
    for ch in input_string:
        print(ch)
```

Lab6StarterHint.py

Summary

Content quiz#3 will take place next week Tuesday, November 4th. Topics are as follows:

- `Sequence: Strings`
- `Sequence: List`
- `Sequence: Tuple`

Assignment#1 will be due on Friday October 31st

Text reading: https://www.py4e.com/html3/04-functions