

Today's Topics:

- Review functions
- If / else / and / not / in

Lee wilkins

Office : 6C.9

Contact: MIO

INTRODUCTION TO COMPUTER PROGRAMMING IN ENGINEERING AND SCIENCE

Comp Sci assignments (All tests and assignments occur Wednesdays in our lab block)

2

Assignment 1: week 4 (10%)

Test 1 week 7 (15%)

Assignment 2 week 8 (10%)

Assignment 3 week 11 (10%)

Test 2 week 13 (15%)

Physics assignments

Assignments (4 x 2%) 8% Date communicated by the Physics teacher

Project 1: Solving differential equations 10% Week 11

Project 2: Applying programming in science 22% Week 15

GRADE BREAKDOWN REVIEW

We can use **try / except** to find errors and handle them appropriately. There are two main types of errors we will focus on for now:

Type Error: This occurs when you're trying to perform an action on the wrong type of data. Ex: trying to do a mathematic operation on a string!

Value Error: The type is correct, but the value is out of range or incorrect. For example, doing `math.sqrt` on a negative number, or trying to convert a string that contains only letters to an int.

TRY / EXCEPT

```
try:  
except ValueError:
```

Try / Except is a way to handle errors. In order to work best, you need to know what type of error you are expecting. If anything in try encounters an error, it will perform what is inside except.

TRY / EXCEPT

```
import math
try:
    math.sqrt(-1)
except ValueError:
    print("cannot square neg num")
```

We can use try / except like this to check for an error and perform an action. Note syntax and indentation.

TRY / EXCEPT

```
import math
try:
    myNum = int(input("Enter a neg number"))
    math.sqrt(myNum)
except ValueError:
    print("cannot square neg num")
    why = input("Why did you do that ?!!")
```

This raises a value error because while it is a number, it is out of range for `math.sqrt` which only takes negative numbers. This is useful here because we don't know what the user will input.

VALUE ERROR

```
my_var = "hello"  
print(my_var + 4)  
TypeError: can only concatenate str (not "int") to str
```

This causes a type error, we can't add a str and an int! We can use an exception here to catch the type error raised.

```
try:  
    my_var = "hello"  
    print(my_var + 4)  
except TypeError:  
    print("can't add an int to a string")  
    my_var = input("Please enter a new number")
```

This raises a type error because the data type is wrong for the operation we want to perform.

TYPE ERROR

Docstrings are a way of describing and documenting your functions so that people can use them later. It is helpful so that your function is easily understandable at a glance.

Description: Simply describing your function.

Arguments/ Params: List each param by name and what it contains

Returns: What/if your function returns anything, what data type it is

DOCSTRINGS

Surround with triple double quotes

Describe the function

List all the arguments / parameters it takes and what they do

```
def my_func(a_string):  
    """To process a string  
    Args:  
    a_string: the string to be interpreted  
  
    Returns:  
    The interpreted string  
    """  
    interp = a_string.count('a')  
    return interp  
print(my_func("aabbbaa"))
```

List what it returns

End with triple double quotes

DOCSTRINGS

```
""" <Description>  
Args:
```

```
Returns:  
"""
```

Follow this format for writing doc strings. If there are no arguments and no returns, follow this format:

```
""" <Description> """
```

DOCSTRINGS

REVIEW

def means we
are defining the
function

This is your
function name

Note the syntax,
you always need
() so you know its
a function

```
def my_function():  
    print("this is my function")
```

Function content is indented

Your function ends, here, but it wont be called until you do this:

```
my_function()
```

This is where your
function actually runs.
This is called **calling the
function**.

FUNCTION SYNTAX

I can pass a parameter through my function. It will be used by this name only within the scope of the function.

Whatever I pass through the function will be placed here

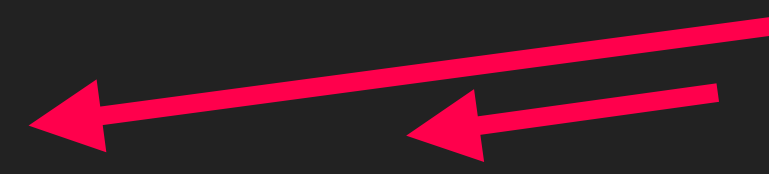
```
def my_function_param(my_string):  
    print(my_string)  
  
my_function_param("this is a string")  
my_function_param("I can print anything here")  
my_function_param("This function is so useful!")
```

PASSING PARAMETERS

We can call the same function to produce different results

You can create variables inside your function. They only exist inside the function and not in the rest of your program

You can pass multiple parameters through your function



```
def calculate_square_area(width, height):  
    area = width * height  
    print(f'The area of a square with the width of {width} inches and the height of {height} inches is {area} inches^2')
```

```
calculate_square_area(2,4)  
# The area of a square with the width of 2 inches and the height of 4 inches is 8 inches^2  
calculate_square_area(5,10)  
# The area of a square with the width of 5 inches and the height of 10 inches is 50 inches^2  
calculate_square_area(100,3)  
# The area of a square with the width of 100 inches and the height of 3 inches is 300 inches^2
```

MULTIPLE PARAMETERS

This function takes no params

```
def my_input_function():  
    take_my_number = int(input("What is the number? "))  
    what_to_add = int(input("What do I add to it? "))  
    print(take_my_number + what_to_add)
```

```
my_input_function()
```

Inputs can be requested
inside the function

INPUT

```
def return_calculate_square_area(width, height):  
    area = width * height  
    return(area)
```



```
print(return_calculate_square_area(2, 4))
```



Return gives us back the
result

We can print return here.

RETURN


```
def return_calculate_square_area_return(width, height):  
    area = width * height  
    return(area)
```



Return gives us back the number. Functions end when they return.

```
print(return_calculate_square_area_return(2,4))
```

```
def return_calculate_square_area_print(width, height):  
    area = width * height  
    print(area)
```



Print only shows us the number. Print can happen many times in a function.

```
return_calculate_square_area_print(2,4)
```

Think of print like showing the user the information, but return is where you give back the number in order to perform another operation on it. For example, `math.sqrt` returns the result, it does not print it unless you choose to print it.

RETURN VS PRINT

Booleans are data types that are either True or False.

They are similar to int, float, string, in that they are a type of information@

Note the capital T and F!

```
my_boolean = True;
```

```
print(my_boolean)
```

```
if my_boolean: my_boolean = False
```

```
print(my_boolean)
```

BOOLEANS

```
myNum = int(input("Enter a number: "))  
myNum2 = int(input("Enter another number: "))
```

```
my_boolean = myNum > myNum2
```

Boolean values can be assigned as the result of a comparison like this. In this example, my_boolean will be true or false. We can print it and see based on the answers.

BOOLEANS

If statements check to see if your statement is true. If the result of the comparison²⁰ is true, then the commands in the if statement are executed.

```
my_var = True
if my_var == True:
    print("its true")
```

```
my_int = 4
if my_int > 2:
    print(f'{my_int} is greater than 2')
```

```
if "hello" in my_string:
    print("Hello is in the string")
```

IF

I can use an else command if my statement is not true. So, if anything else is true, it will be executed.

```
my_string = " This is an example, it doesn't contain the secret word."
if "hello" in my_string:
    print("Hello is in the string")
else: # If anything else is true, else will be executed.
    print("The string doesn't contain hello")
```

ELSE

```
my_string = " This is an example, it doesn't contain the secret word."
```

```
if "hello" in my_string:  
    print("I am looking for hello in the string, and I found it")  
else:  
    print("I am looking for hello in the string, and I didn't find it")
```

```
if "hello" not in my_string:  
    print("I am looking for a string that DOESNT contain hello, and I found  
hello")  
else:  
    print("I am looking for a string that DOESNT contain hello, and I didn't  
find hello ")
```

I can use the keyword not in in order to check if something is NOT IN a string

NOT

We can use an AND or OR chain in our if statements to compare two sets.

In this case, we are comparing the same two statements but with logical and or OR

```
my_bool = False
my_int = 4
if my_int > 2 and my_bool == True :
    print("Both ARE true")
else:
    print("Both are NOT") # This one will run

if my_int > 2 or my_bool == True :
    print("only one has to be true for this to happen") #this one will run i
else:
    print("Neither are true")
```

AND, OR

We can chain multiple if statements with an elif statement. Each elif must have a condition. Else statements do not have conditions.

These will be **executed in order** (top to bottom), but **only one will ever run**.

```
temp = 40
if temp >= 30:
    print("its real warm out")
elif temp >= 20:
    print("its a bit warm")
elif temp >= 10:
    print('getting cold!')
else:
    print("so cold!!!")
```

ELIF

If we don't use an else statement, all if statements will all be tested.

In this example, all of the statements are true, and therefor all get executed! For this example, we don't want that because we only want one to run, so this won't work.

```
temp = 40
if temp >= 30:
    print("its real warm out")
if temp >= 20:
    print("its a bit warm")
if temp >= 10:
    print('getting cold!')
```

IF WITHOUT THE ELSE

However, in this example we **WANT** to check each example to check if the person has too many animals. We want each if statement to run.

```
def animal_limits():  
    number_of_birds = 10  
    number_of_cats = 30  
    number_of_hamsters = 20  
    animal_threshold = 15  
    if number_of_birds > animal_threshold:  
        print("you have too many birds")  
    if number_of_cats > animal_threshold:  
        print("you have too many cats")  
    if number_of_hamsters > animal_threshold:  
        print("you have too many hamsters")
```

```
animal_limits()
```

IF WITHOUT THE ELSE

We can make this into a function called `weather_function()` that asks

```
def weather_function():  
    temp = int(input("What is the temp?"))  
    if temp >= 30:  
        print("its real warm out")  
    elif temp >= 20:  
        print("its a bit warm")  
    elif temp >= 10:  
        print('getting cold!')  
    else:  
        print("so cold!!!")
```

Now call the function:

```
weather_function()
```

MAKING IT A FUNCTION

```
def student_cookies():  
    num_students = int(input("How many students?"))  
    num_cookies = int(input("how many cookies?"))  
    students_are_good = input("Are the students good? y/n")  
    if num_cookies >= num_students:  
        print("there is enough cookies")  
        if "y" in students_are_good.lower():  
            print("and they are good, so they get cookies")  
        else:  
            print("but the students aren\'t good, so no cookies!")  
    else:  
        print("there are not enough cookies :(")  
  
student_cookies()
```

I can use an if statement inside of an if statement in order to make complex decisions. This is where its a good time to use pseudo code or flow charts to understand your problem!

NESTED IF

```
def student_cookies():  
    try:  
        num_students = int(input("How many students?"))  
        num_cookies = int(input("how many cookies?"))  
        students_are_good = input("Are the students good? y/n")  
        if num_cookies >= num_students:  
            print("there is enough cookies")  
            if "y" in students_are_good.lower():  
                print("and they are good, so they get cookies")  
            else:  
                print("but the students aren't good, so no cookies!")  
        else:  
            print("there are not enough cookies :(")  
    except ValueError:  
        print("gotta be the right data type")  
  
student_cookies()
```

More next week: we can catch errors and perform tasks based on them. In this example, if the user inputs the wrong values, we get an error.

WITH VALUE ERRORS

LAB TIME

Coming up: Review, loops, advanced selection statements

Next week (wednesday) in class assignment (graded)

NEXT CLASS: