**Conditionals: if, if-else**

**Learning Objectives:** By the end of this lab you should:

- be comfortable using if-else constructs
- understand relational operators

**Prerequisites:** Before starting this lab, you should be able to:

- define functions
- call functions
- pass arguments/parameters into functions
- return values from functions

---

**Task 1: f(x) (pair task)**

For this task, write a Python function to implement the function \( f(x) \) in the IDLE Shell window. Your code can use a straightforward application of multiple if and return statements or a combination of one if and one if-else statement and multiple return statements (we’ll learn if-elif-else next week).

\[
f(x) = \begin{cases} 
  x - 1 & \text{if } x < 0 \\
  0 & \text{if } x = 0 \\
  x + 1 & \text{if } x > 0 
\end{cases}
\]

The function \( f(x) \) should return one less than the argument when the argument is negative; it should return one more than the argument when the argument is positive; and, it should return zero when the argument is zero. The following demonstrates proper behavior of this function:

```python
>>> f(-3.141)
-4.141
>>> f(0)
0
>>> f(11.111)
12.111
```

After your function is working properly, demonstrate it to your TA to get credit.

---

**Task 2: Leap Year—or Not (pair task)**

A year can be a leap year, a year when there are 29 days in February, if it’s evenly divisible by 4. However, not all years divisible by 4 are leap years. If a year divisible by 4 is a century year, i.e.,
divisible by 100, then it must be divisible by 400 to be a leap year. Thus, 1700, 1800, and 1900, were not leap years, but 1600 and 2000 were.

For this task, write a function in an IDLE Editor window called `leap_year()` that takes a single integer argument, a year, and returns `True` if the year is a leap year and `False` if it isn’t. Use the modulo function (%) in your code to test for divisibility. For example, to test whether a number `num` is evenly divisible by `n`, one can use a statement such as the following:

```python
if num % n == 0:
    print(num, 'is divisible by', n)
```

To test whether a number isn’t divisible by a given value, various options can be used. Here are two examples:

```python
if num % n != 0:
    print(num, 'is not divisible by', n)
```

or, as we’ll learn next week,

```python
if not num % n == 0:
    print(num, 'is not divisible by', n)
```

You can combine the tests for divisibility and indivisibility (because they’re mutually exclusive—a number is either divisible by a given value or it isn’t):

```python
if num % n == 0:
    print(num, 'is divisible by', n)
else:
    print(num, 'is not divisible by', n)
```

There are a number of valid ways to code this problem (and most other ones as well). However, you’ll need to use three modulo statements.

Next write a `main()` function that prompts the user for a year and prints the results of calling `leap_year()` as shown below:

```python
Enter a year [4 digits]: 1988
1988 is a leap year!

Enter a year [4 digits]: 2016
2016 is a leap year!

Enter a year [4 digits]: 1960
1960 is a leap year!

Enter a year [4 digits]: 2014
2014 isn’t a leap year.
```

After your code is working properly, demonstrate it to your TA to get credit.
**Task 3:** Doomsday Algorithm, Take 4 (pair task)

Recall that a year’s Doomsday is given by the equation:

$$Doomsday = (((yr \div 12) + (yr \% 12) + (yr \% 12) \div 4) \% 7 + anchor) \% 7$$  \hspace{1cm} (1)

where $yr$ is the last two digits of the year and the anchor day is an integer between 0 and 7 with 0 corresponding to Sunday and 7 to Saturday.

Previously we wrote a program to calculate a year’s Doomsday that included the function `calc_doom()`. We then added the ability to enter a full, four-digit year in one of our programming assignments. Now we’re going to extend the algorithm so that, assuming we’re only considering 1900-2099, we don’t need to enter the anchor day. Instead we use the first two digits of the year to determine the anchor number. Thus, in this task,

- Write a function `set_anchor()` that takes as its parameter the first two digits of the year and returns a 2 if the digits are 20 and a 3 if the digits are 19. Use multiple `if` and `return` statements.
- Write a function `main()` that prompts the user for a four-digit year (an integer), extracts the first two and last two digits of the year, calls `set_anchor` which returns the anchor, calls `calc_doom()`, and prints out the Doomsday.
- Use the function `calc_doom()` that you wrote before. You should be able to copy and paste it from one Editor window into another.

Anchor days for 2000-2099 are Tuesday (=2) and for 1900-1999 Wednesday (=3). Example results are shown below.

```
1 Enter the year in four digits: 1966
2 Doomsday = 1

4 Enter the year in four digits: 2017
5 Doomsday = 2
```

Run the program for your TA to get credit.

**Task 4:** BMI Again (pair task)

We’ve also seen the BMI several times and created a number of functions to calculate the BMI given some user input. However, we never really evaluated what the BMI means. In this task, we’ll change this. The body mass index (BMI) is given by the formula

$$\text{BMI} = \frac{weight}{(height)^2}$$

where $weight$ is weight (or, more precisely, mass) in kilograms and $height$ is height in meters. We have the following functions from Lab #3:
• `convert_height()` with parameter `height` that converts height to meters

• `convert_weight()` with parameter `weight` that converts weight to kilograms (i.e., mass)

• `calc_bmi()` with parameters `height_m` and `mass`

In this task, you’ll write two more functions:

• `main()`: Prompts a user for his or her weight in pounds and height in inches, calls all other functions as needed, prints the final results.

• `classify_bmi`: Given a BMI, this function will report back a classification to be printed by `main()`.

Also declare global variables `kg_per_lb = 0.45359237` and `m_per_inch = 0.0254` at the beginning of the program, i.e., before defining functions and end with a call to `main()`. The table below gives the BMI classifications:

<table>
<thead>
<tr>
<th>BMI</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.5 or less</td>
<td>Underweight</td>
</tr>
<tr>
<td>18.5 to 24.99</td>
<td>Normal Weight</td>
</tr>
<tr>
<td>25 to 29.99</td>
<td>Overweight</td>
</tr>
<tr>
<td>30 to 34.99</td>
<td>Obesity (I)</td>
</tr>
<tr>
<td>35 to 39.99</td>
<td>Obesity (II)</td>
</tr>
<tr>
<td>40 or greater</td>
<td>Morbid Obesity</td>
</tr>
</tbody>
</table>

1  Enter height [inches]: 72
2  Enter weight [pounds]: 160
3
4  BMI = 21.6996783839
5  Mass = 72.5747792 [g]
6  Height = 1.8288 [m]
7  BMI Classification = Normal Weight

Another example (this time with float inputs):

1  Enter height [inches]: 66.5
2  Enter weight [pounds]: 111.5
3
4  BMI = 17.72678119277855
5  Mass = 50.575549255000006 [kg]
6  Height = 1.6890999999999998 [m]
7  BMI Classification = Underweight

After your program is working properly, show it to your TA and demonstrate that it does what it should.