# MISE Summer Programming Camp

## Class 3 Worksheet - Repetition, Repetition, Repetition

This worksheet has 3 sections, which should take you around 1 hour to complete. Section 1 contains a series of short answer questions about what we did in class. You'll have to run some code in your IDE to see the result and write it down. Section 2 will introduce a new topic we haven't seen in class and so it is optional. You'll be guided through it since it doesn't assume knowledge of the topic. Finally on Section 3 you'll have to write some programs and submit them to CodeForces (**this section is the most important and you should try to solve it!**)

## 1. Review questions

1. What values are in these ranges?
    a. range(4)
    b. range(0, 3)
    c. range(1, 9, 2)
    d. range(4, 0)
    e. range(4, 0, -2)

2. What gets printed?
    a. Code:
```
i = 3
sum = 0
while i < 5:
    sum = sum + i
    i = i + 1
print(sum)
```
    b. Code:
```
i = 0
sum = 0
while i < 10:
    sum = sum + i
    if i == 5:
        continue
    i = i + 1
print(sum)
```
    c. Code:
```
i = 0
sum = 0
while i < 10:
```

```
        sum = sum + i
        if i == 5:
           break
        i = i + 1
    print(sum)
```

3. In your IDE or on an online IDE, write for loops that print out what 2a and 2b print out.

4. What happens in this code? Try guessing what will get printed out first before trying it out in an IDE. If the answer doesn't make sense to you, ask on piazza! It's important that you understand how this code works.

```
sum = 0
for i in range(10):
    if i % 2 == 0:
        continue
    sum = sum + i
    if i % 5 == 0:
        break
print(sum)
```

# 2. Importing Python libraries (optional)

## Basic libraries

So far you have learned that if you define a function, you can use the function in other places in your code. This is really useful for several reasons: you can avoid repeating code by calling a single function in multiple places, and it helps you organize your code into sections that each have their own objectives to do.

But what if you want to use code written by other people? You want to avoid re-inventing the wheel if you can help it, and importing packages is one of the most important techniques we use to do just that.

For this section, you should be programming in Visual Studio or a similar local IDE. For set-up instructions please see the instructions in the previous class worksheets.

Let's say I ask you to implement the floor function, which returns the largest integer smaller than a given number. You may write something like this:

```
pi = 3.1415926
def floor(x):
  if x >= 0:
```

```
    return int(x)
  # x is negative
  if int(x) == x:
    return x
  return int(x) - 1

print(floor(pi))
```

This isn't the prettiest code and it'd be really annoying to have to re-implement it every time. It turns out that you don't have to! Now try typing this code into your IDE:

```
import math
pi = 3.1415926
print(math.floor(pi))
```

What we did was import a third-party package (**math**) and call a function (**floor**) implemented inside of that package. We make our code much more concise, readable, and bug-free by using an existing implementation of the floor function. Check out this link for more detailed documentation on the **math** package: https://docs.python.org/3/library/math.html

Try running some of these. What do you get? (Make sure to **import math** first).

a. **print(math.floor(100.2))**
b. **print(math.floor(-0.5))**
c. **print(math.factorial(3))**
d. **print(math.ceil(-0.5))**

In addition to importing functions from a package, we can also import constants. For example, the **math** package already defines a **pi** constant. Try printing out some of these:

a. **print(math.pi)**
b. **print(math.sin(math.pi/2))**
c. **print(math.sin(math.pi))** (Discussion: why is this not 0?)
d. **print(math.inf)**
e. **print(math.log(math.inf))**
f. **print(1/math.inf)**

## More on Import Syntax

There are a couple ways to import things from a package. You will often see both of these ways to import and use objects from a package:

```
import math
print(math.floor(math.pi))
```

```
from math import floor, pi
print(floor(pi))
```

An asterisk (*) is also often used in programming as a symbol that means "everything". So you may also sometimes see this code, even though it is discouraged because it can lead to confusing code.

```
from math import *
print(floor(pi))
```

## More packages

**math** is such a common package that it's included with most python distributions. Some other common packages for which this is true are **os** and **sys**. For less common packages, you cannot directly import the package in your code, you also need to install the package on your computer in order to be able to import them.

This is true of a lot of commonly used packages like **pytest** (for writing tests to evaluate your code) and **numpy** (for data analysis).

The easiest way to install packages is to use a Python package manager. You might need to do this later in this class, but even now it will be helpful as you write more complicated programs. Here are optional installation instructions if you ever find yourself needing them:

1. Install a python package manager. I would recommend using pip (https://pip.pypa.io/en/stable/installing/). Follow the instructions under the Installing with get-pip.py section
2. Now open VSCode. Open the Terminal (from the top select Terminal > New Terminal).
3. Install the **emoji** package. On Windows this is **py -m pip install emoji**. See this link for more details: https://pip.pypa.io/en/stable/user_guide/
4. Now in your Python code you should be able to write **import emoji** without getting an **ImportError**.
5. Try running the following code in VSCode:

   ```
   import emoji
   print(emoji.emojize(":thumbs_up:"))
   ```

   You should see 👍 in your std output. If you have gotten this far, congratulations, you can easily install and import any python package you want!

## 3. Coding questions

You should see a new problem set in codeforces called “Class 3 Problems” (https://codeforces.com/group/K1Fxw6skwV/contest/443304). There are 6 problems available, but only the first 2 (“For the Collatzive Good” and “Prime Numbers”) are mandatory. The next 2 (“Most Common Letter” and “Parentheses Matching”) are harder and optional, so if you feel up to it you can give it a go, but you don’t have to. The final 2 (“Blank Space” and “Find and Replace”) are even harder, so you should only try to solve them if you want an extra challenge.