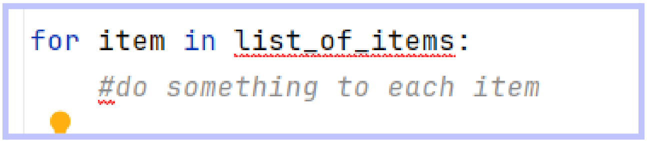
DAY 5: Python Loops

17/01/2025

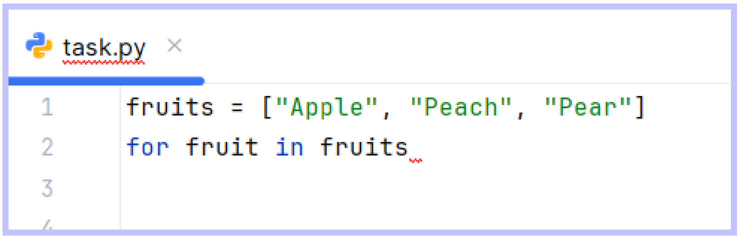
The first concept that we need to know about is loops, things that have to happen over and over and over again and there's types:

For Loop



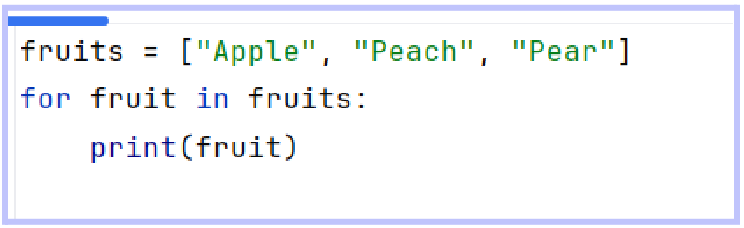
```python
for item in list_of_items:
    #do something to each item
```

It can be used really easily with lists. By using a for loop like this, we can go through each item in a list and perform some action with each individual item. Lets try out in practice:



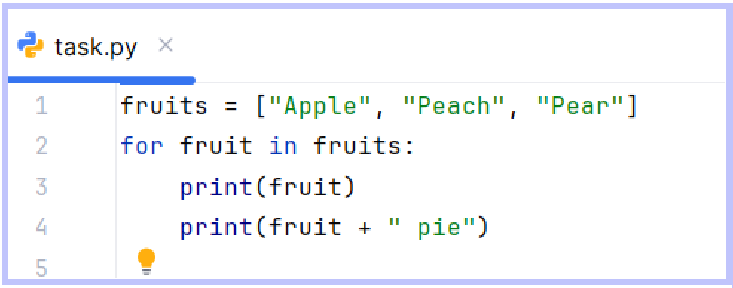
```python
fruits = ["Apple", "Peach", "Pear"]
for fruit in fruits
```

If we wanted to access each item in this list individually and print it out one by one, then we would use a for loop. So we start out with the keyword "for" and then we give a name to a single item, so in this case we might call it fruit. And then we use the "in" keyword. Then finally we print the list that we wanna loop through, which is our fruits



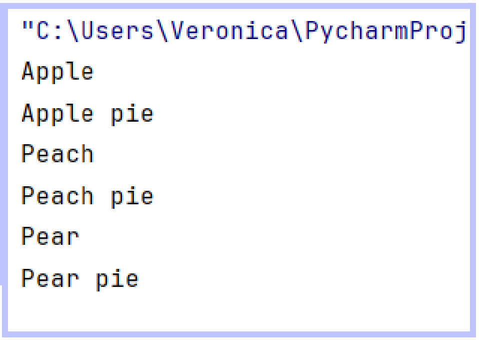
```python
fruits = ["Apple", "Peach", "Pear"]
for fruit in fruits:
    print(fruit)
```

And if I run this code you'll see that it loops through my list of fruits, and for each of the fruits inside the list, it prints it out into the console. And this really emphasizes the most important aspect of loops. The loop allows us to execute the same line of code multiple times. In this case we're executing the print statement three times But our for loop isn't limited to just executing a single statement. We don't just have to print out the name of an item in the list. We can execute a whole block of statements multiple times, and we can do many things inside this for loop, and by inside I mean indented. So let's say that in addition to printing out the fruit, I'm going to write another print statement, and I'm going to not only print out the name of the fruit, but I'm also going to say fruit + space plus pie. So when I run this code



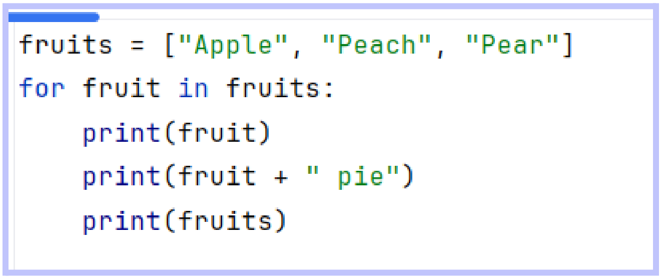
task.py

```python
fruits = ["Apple", "Peach", "Pear"]
for fruit in fruits:
    print(fruit)
    print(fruit + " pie")
```



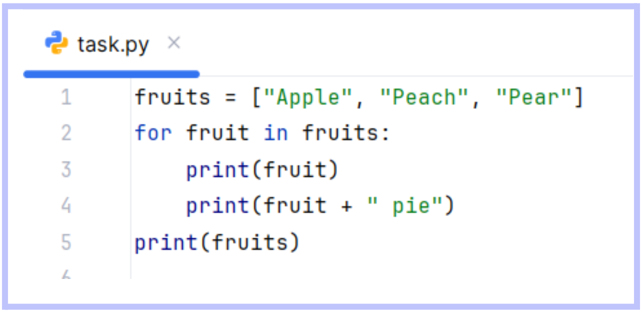
```
"C:\Users\Veronica\PycharmProj
Apple
Apple pie
Peach
Peach pie
Pear
Pear pie
```

So this is how we can implement a simple for loop that loops through a list and assigns a variable name to each of the items in the list in order, and then inside the for loop after the colon, after some indentation, we can do something with that temporary variable for each of the items.
Whenever you see a colon, say in our if statements that we saw previously, or the for loop that we're using here, the indentation is really, really important because if it's indented then it means that it's inside the for loop and these instructions will get carried out for as many times as the for loop will need to repeat.



```python
fruits = ["Apple", "Peach", "Pear"]
for fruit in fruits:
    print(fruit)
    print(fruit + " pie")
    print(fruits)
```

Now, if I had indented that back to the beginning, so now it's no longer inside the for loop, then it's only going to print once and it's going to print it after the for loop is done.



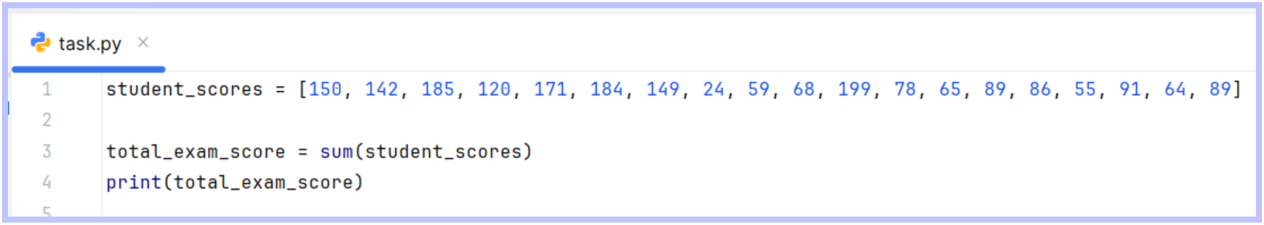
task.py

```python
fruits = ["Apple", "Peach", "Pear"]
for fruit in fruits:
    print(fruit)
    print(fruit + " pie")
print(fruits)
```

So the indentation is really, really important, and you have to be really careful with this.

Highest Score

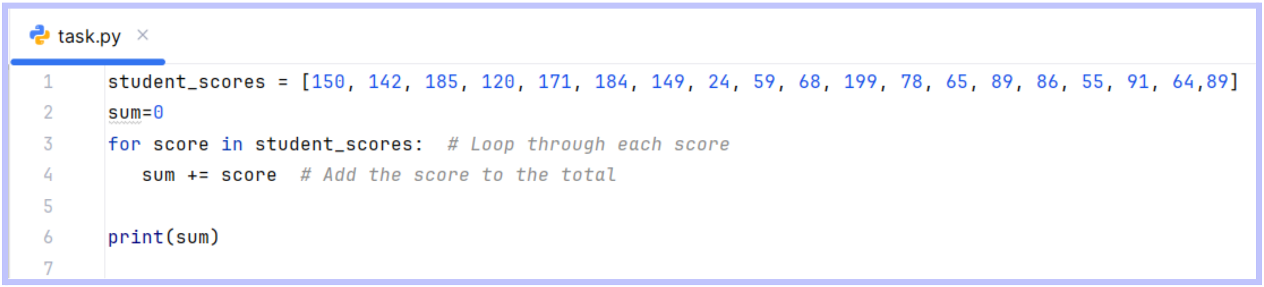
Python is extremely number friendly, It has a lot of built in functions to help us work with numbers. For example, if I had a class of students and I listed each of their exam scores inside this giant list, and I wanted to work out, well, what is the total exam score. Python has something called the sum() which allows us to put in any iterable data type, including lists. So I can put the student scores into here.



```python
student_scores = [150, 142, 185, 120, 171, 184, 149, 24, 59, 68, 199, 78, 65, 89, 86, 55, 91, 64, 89]

total_exam_score = sum(student_scores)
print(total_exam_score)
```

The output equals to 2068 now if I wanted to do this manually though, I could because at the end of the day, all that's happened is the Python team, when they developed the programming language they created each of these helpful functions so we could create some ourselves as well, given that we understand how for loops work. So let's create a variable called sum and set it to start off as equal to 0. And then we could say, sum += score.

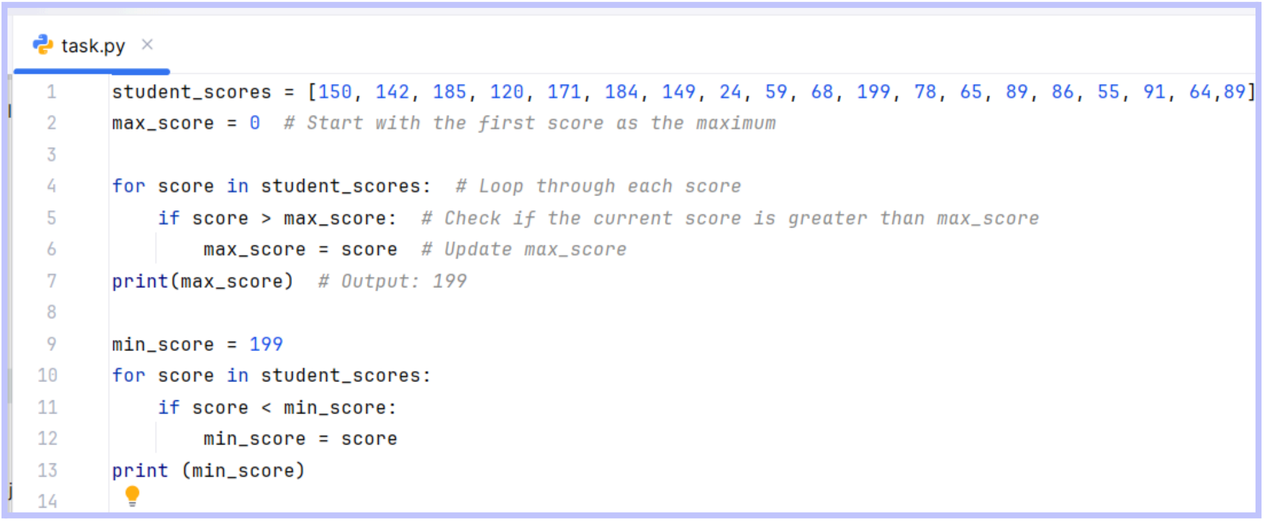


```python
student_scores = [150, 142, 185, 120, 171, 184, 149, 24, 59, 68, 199, 78, 65, 89, 86, 55, 91, 64,89]
sum=0
for score in student_scores:  # Loop through each score
   sum += score  # Add the score to the total

print(sum)
```

And every time this for loop runs it's going to go to the next score and the next score, and each time that score is added to the previous value of sum, so then it accumulates in that variable.

Challenge #1



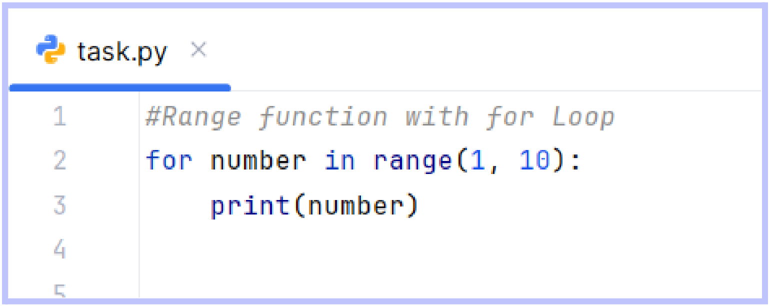
```python
student_scores = [150, 142, 185, 120, 171, 184, 149, 24, 59, 68, 199, 78, 65, 89, 86, 55, 91, 64,89]
max_score = 0  # Start with the first score as the maximum

for score in student_scores:  # Loop through each score
    if score > max_score:  # Check if the current score is greater than max_score
        max_score = score  # Update max_score
print(max_score)  # Output: 199

min_score = 199
for score in student_scores:
    if score < min_score:
        min_score = score
print (min_score)
```

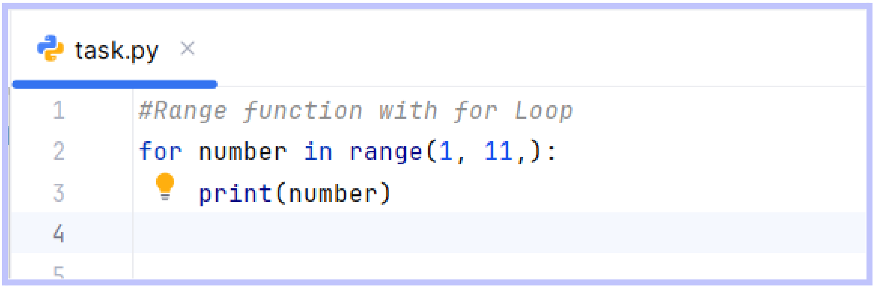
For Loops and the range () Function

We've only using for loops in association with lists. But we're not always going to be working with lists and using for loops.
A good example is Carlus Gauss when he was ten years old, his math teacher gave him an exercise that she probably thought would tie him up for a little while, and the idea was to get him to add all of the numbers from 1 to 100. So 1 + 2 +3 + 4 + 5 et cetera, all the way until 100. Well, he is actually a really smart kid, and he figured out that if you flip the numbers around. So 100 plus 99 plus 98, and you look at both of these two lines, you can see that 1 + 100 = 101 2 + 99 = 101, 3 + 98 = 101. Basically, if you tie all of these numbers together, there's actually 50 pairs of 101. So he could simply just do 50 multiplied by 101 which is 5050. We can use the Range() in order to do this, Lets see this in code:



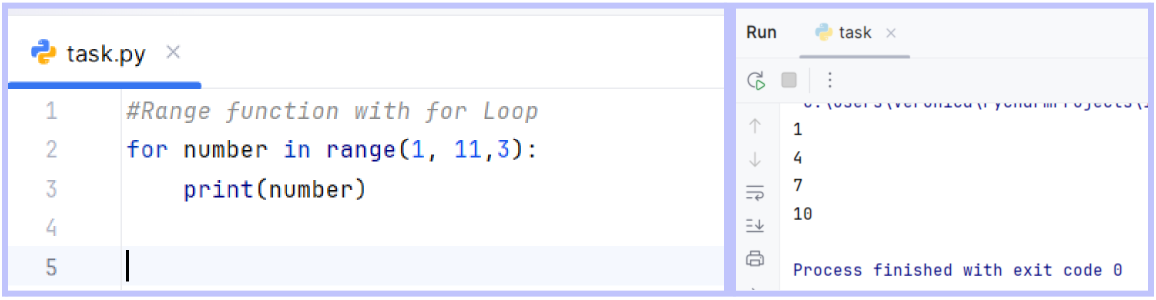
```python
#Range function with for Loop
for number in range(1, 10):
    print(number)
```

So for example if I wrote, "for number in range..." and then my range is going to be (1, 10). So this is going to be the range that I'm going to create between 1 and 10, and then I want to get hold of each of the numbers inside that range. And this is going to be between 1 and 10 and not including ten.
So if I wanted all of the numbers from 1 to 10, I actually have to set a range between 1 and 11.



```python
#Range function with for Loop
for number in range(1, 11,):
    print(number)
```

Now by default the range() function will step through all the numbers from the start to the end, and it will increase by one. Now, if you wanted to increase by any other number, then you have to add another comma to the end of it and specify how large you want the step to be. So let's say we change the step size to 3.



```python
#Range function with for Loop
for number in range(1, 11,3):
    print(number)
```

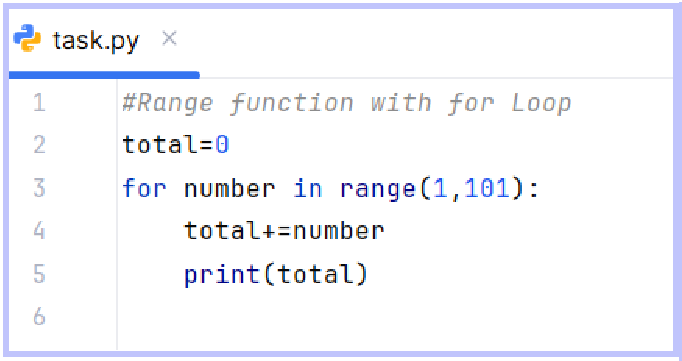
```
1
4
7
10

Process finished with exit code 0
```

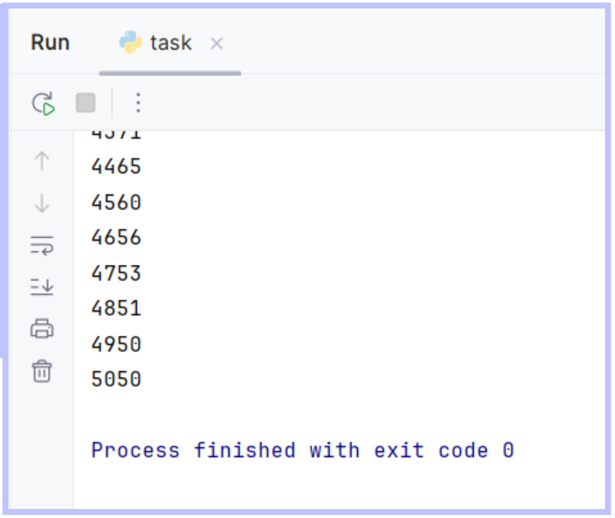
Now if I rerun the same code, you'll see that it goes from 1, and then it steps by 3 to 4, and then it steps by 3, and then it steps by 3, finally to 10.

Challenge #2

Now let's come back to the problem that I mentioned at the beginning of this lesson,
How can we add up all of the numbers from 1 to 100 by using code? See if you can solve the Gauss challenge.



```python
#Range function with for Loop
total=0
for number in range(1,101):
    total+=number
    print(total)
```



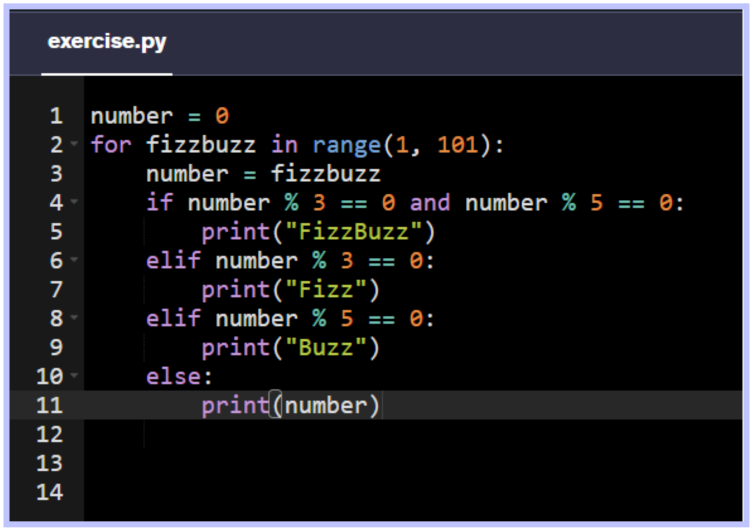
```
4371
4465
4560
4656
4753
4851
4950
5050

Process finished with exit code 0
```

Challenge #3: FizzBuzz

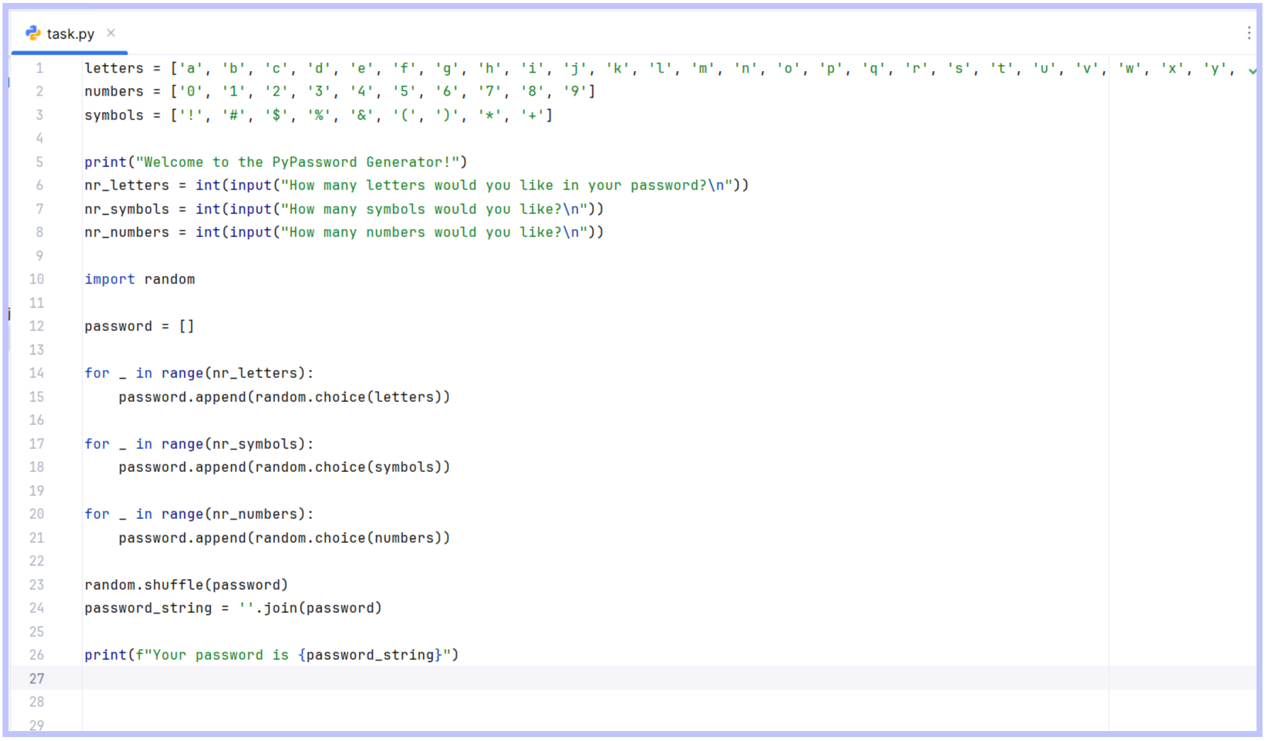
You are going to write a program that automatically prints the solution to the FizzBuzz game. These are the rules of the FizzBuzz game:

- Your program should print each number from 1 to 100 in turn and include number 100.
- But when the number is divisible by 3 then instead of printing the number it should print "Fizz".
- When the number is divisible by 5, then instead of printing the number it should print "Buzz".`
- And if the number is divisible by both 3 and 5 e.g. 15 then instead of the number it should print "FizzBuzz"



```python
number = 0
for fizzbuzz in range(1, 101):
    number = fizzbuzz
    if number % 3 == 0 and number % 5 == 0:
        print("FizzBuzz")
    elif number % 3 == 0:
        print("Fizz")
    elif number % 5 == 0:
        print("Buzz")
    else:
        print(number)
```

Project: Password Generator Project



task.py

```python
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y',
numbers = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
symbols = ['!', '#', '$', '%', '&', '(', ')', '*', '+']

print("Welcome to the PyPassword Generator!")
nr_letters = int(input("How many letters would you like in your password?\n"))
nr_symbols = int(input("How many symbols would you like?\n"))
nr_numbers = int(input("How many numbers would you like?\n"))

import random

password = []

for _ in range(nr_letters):
    password.append(random.choice(letters))

for _ in range(nr_symbols):
    password.append(random.choice(symbols))

for _ in range(nr_numbers):
    password.append(random.choice(numbers))

random.shuffle(password)
password_string = ''.join(password)

print(f"Your password is {password_string}")
```