01 Python for Everyone Except Grandma; Sorry Grandma

1. Flow Control Statement

If Statement

```
age = 18
if age >= 18:
    print("You are eligible to vote")
```

- The if statement is used to execute a block of code only if the condition specified in the statement is true.
- The code inside the if block will only be executed if the condition specified is true.
- The condition in the if statement can be any expression that returns a boolean value.
- The if statement can be followed by an optional else block which will be executed if the condition in the if statement is false.
- You can use multiple elif statements to check for multiple conditions.

Else Statement

```
for i in range(10):
    if i == 5:
        print("Encountered the number 5!")
        break
else:
    print("Finished iterating over the loop without encountering the number 5")
```

- The else statement can be used in conjunction with if, for, or while statements. When used with a for or while loop, the else block will be executed if the loop completes all its iterations without encountering a break statement. When used with an if statement, the else block will be executed if the condition in the if statement is false.
- The else block is optional and can only be used after an if, for, or while statement.
- When used with a for or while loop, the else block will be executed if the loop completes all iterations without encountering a break statement.
- When used with an if statement, the else block will be executed if the condition in the if statement is false.
- The else block will not be executed if the loop is exited prematurely by a break statement.

Elif Statement

```
age = 15
if age < 18:
    print("You are not old enough to vote")
elif age == 18:
    print("You just became eligible to vote")
else:
    print("You are eligible to vote")
```

- You can use as many elif statements as you need in order to test multiple conditions.
- The elif block is optional, you can use only the if statement or combine it with the else statement.
- The elif statement must be used after an if statement and before the else statement (if present).
- If multiple conditions are true, only the block of code belonging to the first true condition will be executed.

For statement

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

- The for loop will iterate over the sequence of items and execute the block of code for each item.
- You can use the range function to generate a sequence of numbers to iterate over.
- You can use the break and continue statements inside the for loop just like in the while loop.
- You can use the else block to execute code when the for loop finishes iterating over the sequence.

While statement

```
count = 0
while count < 5:
    print(count)
    count += 1
```

- The while loop is used to repeatedly execute a block of code as long as the condition specified in the statement is true.
- The while loop will keep looping as long as the condition specified is true.
- Be careful not to create an infinite loop by making sure the condition will eventually become false.
- You can use the break statement to exit the loop prematurely.
- You can use the continue statement to skip over certain iterations of the loop.

---

2. Function with parameter

```
def greet(name):
    print("Hello, " + name + "!")

greet("Alice")
greet("Bob")
```

- A user-defined function is a block of reusable code that performs a specific task. It is created by the user to fulfill a specific requirement, instead of using built-in functions in the programming language.
- A function with parameters is defined using the `def` keyword, followed by the function name and a list of parameters in parentheses.
- The parameters act as placeholders for the actual values that will be passed to the function when it is called.
- In the example above, the `greet()` function takes a single parameter `name`, which is used to print a personalized greeting.
- When the `greet()` function is called, it expects an argument to be passed in place of the `name` parameter. This value is then used within the function body.
- In the example above, the `greet()` function is called twice with different arguments: `"Alice"` and `"Bob"`. Each call to the function results in a personalized greeting for the given name.
- The `print()` function is used to output the greeting message to the console. This is a common way to communicate with the user of the program.

---

3. Temperature Conversion

```
while True:
    try:
        celsius = float(input("Enter temperature in Celsius: "))
        fahrenheit = (celsius * 9/5) + 32
        print("Temperature in Fahrenheit: ", fahrenheit)
        break
    except ValueError:
        print("Invalid input. Please enter a number.")
```

- In this code, we prompt the user for a temperature in Celsius using the `input()` function. We then convert this temperature to Fahrenheit using the formula `(celsius * 9/5) + 32` and print out the result.
- The `try` block contains the code that we want to execute, and any exceptions that are raised within this block will be caught by the `except` block.
- In this case, we use the `float()` function to convert the user's input to a float value. If the input cannot be converted to a float (e.g. if the user enters a string), a `ValueError` exception will be raised.
- The `except` block specifies the type of exception that we want to catch (in this case, `ValueError`) and the code that we want to execute when the exception is caught (i.e. print an error message and prompt the user to enter a valid input).
- The `break` statement is used to exit the loop and continue with the rest of the code if the user enters a valid input.
- If the user enters an invalid input multiple times, the loop will continue to prompt the user until a valid input is entered.

---

4. Sys Exit

```
import sys

while True:
   print('Type exit to exit.')
   response = input()
   if response == 'exit':
```

```
        sys.exit()
    print('You typed ' + response + '.')
```

1. The code demonstrates an infinite loop that can only be terminated by typing the word "exit" when prompted. The `while True:` statement creates an infinite loop that will continue running until it encounters a `break` statement or the program is interrupted.

2. The `input()` function waits for user input and stores it in the `response` variable. In this case, it prompts the user to type the word "exit" to terminate the program.

3. The `sys.exit()` function is used to terminate the program immediately if the user types "exit". It raises the `SystemExit` exception which can be caught and handled by the calling code if necessary.

4. The code also demonstrates string concatenation using the `+` operator. The statement `print('You typed ' + response + '.')` concatenates the string 'You typed ' with the value stored in `response`, and then appends the period symbol '.' to the end of the resulting string before printing it to the console.

---

5. Guess the Number

```
import random

secretNumber = random.randint(1, 20)
print('I am thinking of a number between 1 and 20.')

# Ask the player to guess 6 times.
for guessesTaken in range(1, 7):
    print('Take a guess.')
    guess = int(input())

    if guess < secretNumber:
        print('Your guess is too low.')
    elif guess > secretNumber:
        print('Your guess is too high.')
    else:
        break  # This condition is the correct guess!

if guess == secretNumber:
    print('Good job! You guessed my number in ' + str(guessesTaken) + ' guesses!')
else:
    print('Nope. The number I was thinking of was ' + str(secretNumber))
```

1. The code generates a random integer between 1 and 20 using the `random.randint()` function, and prompts the user to guess the number in 6 attempts using a `for` loop.

2. The `input()` function waits for user input and stores it in the `guess` variable. In this case, it prompts the user to enter their guess for the random number.

3. The `if` statement checks whether the user's guess is too high or too low compared to the random number, and provides feedback to the user accordingly. If the guess is correct, the `else` statement is executed and the loop is broken using the `break` keyword.

4. The code also demonstrates string concatenation using the `+` operator. The statement `print('Good job! You guessed my number in ' + str(guessesTaken) + ' guesses!')` concatenates the string 'Good job! You guessed my number in ' with the value stored in `guessesTaken`, and then appends the string 'guesses!' to the end of the resulting string before printing it to the console. Additionally, the `str()` function is used to convert an integer to a string.

---

6. Global and Local variables

```
def spam():
    global eggs  # declare eggs as a global variable
    eggs = 'spam'  # set value of global eggs variable

def bacon():
```

```
    eggs = 'bacon'  # declare and set value of local eggs variable
    print(eggs)  # print value of local eggs variable

def ham():
    print(eggs)  # print value of global eggs variable

eggs = 42  # declare and set value of global eggs variable
spam()
bacon()  # prints 'bacon'
ham()  # prints 'spam'
```

1. In this program, we declare `eggs` as a global variable within the `spam` function. When we call `spam`, it sets the value of the global `eggs` variable to `'spam'`. In the `bacon` function, we declare and set the value of a local `eggs` variable. When we call `bacon`, it prints the value of the local `eggs` variable, which is `'bacon'`. In the `ham` function, we print the value of the global `eggs` variable. When we call `ham`, it prints the value of the global `eggs` variable, which is `'spam'`.

2. Finally, outside of any functions, we declare and set the value of the global `eggs` variable to `42`. When we call `spam`, it changes the value of the global `eggs` variable to `'spam'`, and when we call `ham`, it prints the value of the global `eggs` variable, which is now `'spam'`.

3. Local scope in Python is the scope created within a function, where variables and parameters are defined and exist only during the execution of that function. These variables are not accessible outside of that function. Global scope in Python is the scope created outside of all functions, where variables and parameters are defined and exist throughout the program. These variables can be accessed from any function or module in the program.

4. Rules regarding global and local variables

    a. Global code cannot access local variables.
    b. Local code can access global variables.
    c. Local code in one function cannot access variables from another local scope.
    d. Same name can be used for different variables in different scopes.

---

7. Exception Handling

```
def divexp(a, b):
    if b == 0:
        raise ZeroDivisionError("Error: denominator should be non-zero")
    else:
        c = a / b
        return c

a = int(input('Enter first integer: '))
assert a > 0, "Error: number a is negative"
b = int(input('Enter second integer: '))

try:
    print(divexp(a, b))
except ZeroDivisionError as e:
    print(e)
```

1. Exception handling is a programming technique that allows us to detect and handle errors in our code instead of letting the program crash.

2. In Python, we can use try-except blocks to handle exceptions. Code that could potentially raise an exception is placed in the try block, and code to handle the exception is placed in the except block.

3. The except block catches the exception and allows the program to continue running. It can also print an error message or perform some other action to handle the exception.

4. The given code defines a function called `divexp` that takes two arguments, a and b, and returns their quotient. If the denominator b is zero, the function raises a ZeroDivisionError with a custom error message.

5. The code also uses an `assert` statement to check that the input value of a is positive. If it is not, an AssertionError is raised with a custom error message.

6. The code then prompts the user to input two integers and calls the `divexp` function with those inputs.

7. If the function raises a ZeroDivisionError, the code catches it using a try-except block and prints the error message.

---

8. Collatz Sequence

```
def collatz(number):
    if number % 2 == 0:
        result = number // 2
        print(result)
        return result
    else:
        result = 3 * number + 1
        print(result)
        return result
```

1. The program demonstrates the use of conditional statements in Python to differentiate between even and odd numbers.

2. It showcases how to define and call a function in Python, with the function taking a single parameter.

3. The program demonstrates the use of the mathematical Collatz conjecture, which states that for any positive integer, if you repeatedly apply a specific mathematical operation to it (dividing by 2 if even, or multiplying by 3 and adding 1 if odd), you will eventually reach the number 1.

4. The program highlights the importance of proper indentation in Python, as indentation is used to define the scope of the collatz() function and its conditional statements.