USER INPUT AND LOOPS

Essentials of Repetition

Loops

- Repetition statements (or **loops**) repeat an action
- Each repetition of action is known as **pass** or **iteration**
- Two types of loops
  - *Those that repeat action a predefined number of times (**definite iteration**)*
  - *Those that perform action until program determines it needs to stop (**indefinite iteration**)*

The *for* Loops (1 of 4)

- Python's ***for*** loop is the control statement that most easily supports **definite** iteration

```
for var in sequence:
    indented block of statements
```

- The line beginning with *for* is called the **header** of the loop
- The variable following the word *for* is called the **loop variable**
- The indented block of statements is called the **body** of the loop
- Each execution of the body is referred to as a **pass** through the loop

The *for* Loops (2 of 4)

- Example: Print all numbers from 0 to 5, and print a message when the loop has ended:

```
for x in range(6):
    print(x, end = " ")
else:
    print("Finally finished!")
```

```
0 1 2 3 4 5 Finally finished!
```

The *for* Loops (3 of 4)

- Example: Store all squares from 1 to 10 in a list, and print each square number

```
10  squares = [value**2 for value in range(1, 11)]
11  for x in squares:
12      print(x, end = " ")
```

```
1 4 9 16 25 36 49 64 81 100
```

The *for* Loops (4 of 4)

Ex1_for_pass.py

- There are times when you want loop to cycle through a sequence and not do anything
  - *The **pass** statement should be used.*
- The pass statement is a **do-nothing** placeholder statement

```
4 for x in squares:
5       pass
```

The *Input()* Function

Ex2_greeter.py

- Prompts the user to enter data

```
1  prompt = "If you tell us who you are, we can personalize the messages you see."
2  prompt += "\nWhat is your first name? "
3
4  name = input(prompt)
5  print(f"\nHello, {name}!")
```

  - *User types response, presses ENTER key*
  - *Entry assigned to variable on left*
  - *Python interprets everything the user enter as a **string***

Accept Numerical Input

Ex3_rollercoaster.py

- int() returns an integer number
- float() - returns a floating point number
- eval() - evaluates and executes an expression

```
1  height = input("How tall are you, in inches? ")
2  height = int(height)
3
4  if height >= 36:
5      print("\nYou're tall enough to ride!")
6  else:
7      print("\nYou'll be able to ride when you're a little older.")
```

Using Modulo Operator

Ex4_even_or_odd.py

```
1  number = input("Enter a number, and I'll tell you if it's even or odd: ")
2  number = int(number)
3
4  if number % 2 == 0:
5      print(f"\nThe number {number} is even.")
6  else:
7      print(f"\nThe number {number} is odd.")
```

- Try it yourself: Ask the user for an integer number, and then display whether the number is a multiple of 10 or not.

PE7_1 & PE7_2

1. Request an integer input, and then print whether the number is a multiple of 10 or not.

```
Example Output 1
Enter an integer number, and I'll tell you if it's a multiple of ten: 15
15 is not multiple of ten.

Example Output 2
Enter an integer number, and I'll tell you if it's a multiple of ten: 50
50 is a multiple of ten.
```

2. For & While

a) Use a *for* loop to print all the numbers are **even** and **multiples of 3** from 1 to 1000 inclusive.

b) Convert the *for* loop to a *while* loop.

```
Example Output
6 12 18 24 30 36 42 48 54 60 66 … 996
```

The *while* Loop (1 of 2)

- Executes a block of code repeatedly
- ***while*** loop repeatedly executes an indented block of statements
  - *As long as a certain* ***condition*** *is met*

```
while condition:
    indented block of statements
```

- A *while* loop can be used to ensure that a proper response is received from a request for input. This process is called ***input validation***.

The *while* Loop (2 of 2)

```
while condition:
    indented block of statements
```

- The line beginning with *while* is called the **header** of the loop
- The condition in the header is called the **continuation condition** of the loop
- The indented block of code is called the **body** of the loop
- Each execution of the body of a loop is called a **pass**
- The **continuation condition** is a Boolean expression that evaluates to either **True** or **False**

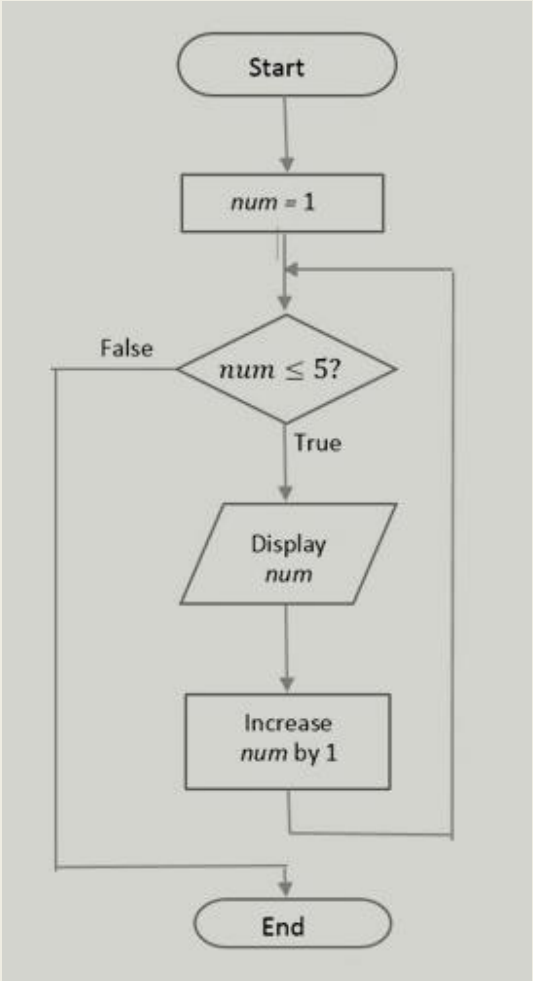
Counter-controlled *while* Loop

Ex5_counting5.py

- Use ***while*** for a counter-controlled loop
- Example: displays 1 - 5,
  after loop terminates, num will be 6

```
1  current_number = 1
2  while current_number <= 5:
3      print(current_number)
4      current_number += 1
5  print(f"current_number = {current_number}")
```

```
1
2
3
4
5
current_number = 6
```



PE7_3

3. Loop & Calculation

a) Use a *for* loop to calculate and print the **sum of all numbers** between 1 to 100 inclusive.

```
Example Output
Sum = 5050
```

b) Use a *while* loop to calculate and print the **sum of all even numbers** between 1 to 100 inclusive.

```
Example Output
Sum = 2550
```

The *while* Loop with *Input*()

Ex6a_while_input_number.py

- Use ***while*** for a counter-controlled loop
- Example: using *input*() to request the counter value

```
1  n = int(input("Enter a number: "))  #1. get n.
2  i = 1;                              #2. initialize i to 1
3  while i <= n :                      #3. check condition
4    print(i)                          #4. repeat action
5    i = i + 1;                        #5. update i
6  print(f"current_number = {i}")
```

```
Enter a number: 5
1
2
3
4
5
current_number = 6
```

PE7_4 & PE7_5

4. Use a loop to print all the numbers are **odd** and **multiples of 5** from 1 to *n* inclusive.
a) n is a user input.
b) Implement (an *if-else)* statements to **validate** n.
Input text can be any content. Just make sure to precisely match the output format below.

```
Example Output 1
Enter a positive number: 65
Range = 1 to 65
5 15 25 35 45 55 65
```

```
Example Output 2
Enter a positive number: 0
Range = 1 to 0
Invalid input.
```

5. While & For
a) Implement a *while* loop to print all the numbers from 9 to 1 inclusive.
b) Then display *Happy New Year!*
c) Convert the *while* loop to a *for* loop

```
Example Output
9
8
7
6
5
4
3
2
1
Happy New Year!
```

The *while* Loop with Lists

Ex6b_pets.py & Ex6c_confirmed_users.py

- Allow collect, store, and organize
- Example: using *in* keyword

```
1  pets = ['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']
2  print(pets)
3
4  while 'cat' in pets:
5      pets.remove('cat')
6
7  print(pets)
```

```
['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']
['dog', 'dog', 'goldfish', 'rabbit']
```

Indefinite Iteration *while* Loop

Ex7_while_input_positive.py

- Use ***while*** for a **sentinel-controlled** loop
- A **sentinel value** is used to indicate the end of a sequence of inputs
- Example: using input() to request the sentinel value

```
1   n = int(input("Enter a positive number: "))
2   while n <= 0:
3       n = int(input("Try again: "))
4   print("You enetered a postive number.")
```

```
Enter a positive number: -1
Try again: 0
Try again: 1
You enetered a postive number.
```

The *break* Statement

Ex8a_while_break.py

- Use ***break*** to stop the loop even if the while condition is true

```
1  i = 1
2  while i < 6:
3      print(i)
4      if i == 3:
5          break  # the loop is immediately exited when i is 3
6      i += 1
```

```
1
2
3
```

while True & *break*

Ex8b_cities.py

- If the loop must run at least once, use a ***while True*** loop with delayed examination of termination condition
- Ensure a **break** statement to be reached eventually

```
prompt = "\nPlease enter the name of a city you have visited:"
prompt += "\n(Enter 'quit' when you are finished.) "

while True:
    city = input(prompt)
    if city == 'quit':
        break
    else:
        print(f"I'd love to go to {city.title()}!")
```

While True & Flag

Ex9_parrot.py

- A **flag** is a variable used to indicate whether a certain event has occurred or a certain situation exists

```
prompt = "\nTell me something, and I will repeat it back to you:"
prompt += "\nEnter 'quit' to end the program. "
active = True
while active:
    message = input(prompt)
    message = message.lower()
    if message == "quit":
        active = False
    else:
        print(message)
```

The *continue* Statement

Ex10a_while_continue.py & Ex10b_counting.py

- Use ***continue*** to stop the current iteration, and jump to the next iteration

```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

```
1
2
4
5
6
```

```
current_number = 0
while current_number < 10:
    current_number += 1
    if current_number % 2 == 0:
        continue
    print(current_number)
```

```
1
3
5
7
9
```

The *else* Statement

Ex11_while_else.py

- *Use* ***else*** *to* run a block of code once when the condition of ***while*** no longer is true

```
1   n = []                  #define a list
2   i = 1                   #define a counter
3   total = 0               #define an accumulator
4   while i < 6:            #validate the continuation condition
5       total+=i            #use the accumulator to hold the sum
6       n.append(i)         #populate the list
7       print(i)            #display each number
8       i += 1              #update counter
9   else:
10      print(f"\nSum = {total}")
11      print(f"Sum = {sum(n)}")
12      print(n, end = " ")
```

```
1
2
3
4
5

Sum = 15
Sum = 15
[1, 2, 3, 4, 5]
```

Avoid Infinite Loops without Exit Conditions

Ex12a_infinite_loop.py & Ex12b_infinite_loop.py

- Every loop needs a way to stop running so it won't continue to run forever

```
1  n = 1
2  while n <= 5:
3      print(n)
```

```
1  n = 1
2  while n <= 5:
3      n -= 1
```

- Test every loop and make sure the loop stops when you expect it to

PE7_6 & PE7_7 & PE7_8 & PE7_9

- Write your codes and run

Summary (1 of 2)

- The parentheses of the ***range*** function can contain one, two, or three values.
- When the parentheses contains two or three values, the **first** value is always the **beginning** of the sequence generated.
- When the parentheses contains a single number, call it *n*, no sequence will be generated when n <= 0; otherwise, the sequence of n numbers from o to *n-1* will be generated.
- The values generated by the ***range*** function can be displayed by applying the ***list*** function. For instance, print(list(range(1, 8, 2)) displays [1, 3, 5, 7].
- The ***range*** function generates an arithmetic progression of numbers.
- A ***for*** loop repeats a block of statements as its loop variable **iterates** through a sequence.

Summary (2 of 2)

- If a ***break*** statement is encountered during a pass through a ***while*** loop, the loop is immediately **exited**.
- If a ***continue*** statement is encountered in the block of a ***while*** loop, execution **jumps** back to the closest enclosing while loop header
- A ***while*** loop might use a **counter variable** to keep track of the number of times a certain event has occurred, and **accumulator variable** to hold a total, and a **sentinel value** to indicate the end of sequence of inputs.
- The statements ***break*** and ***continue*** have the same effect in ***for*** loops as they do in ***while*** loops.
- Any type of loop can be **nested** inside another loop. For example, for loops can be nested inside while loops and vice versa
- The **pass** statement is a **do-nothing** placeholder that is sometimes used where the syntax requires a statement.
- Be careful to avoid **infinite loops**; that is, loops that never end.

Input & Loops - Terminologies

- ❑ 1 Accumulator
- ❑ 2 Break
- ❑ 3 Command Prompt
- ❑ 4 Continue
- ❑ 5 Continuation Condition
- ❑ 6 Counter
- ❑ 7 Definite iteration
- ❑ 8 Exit
- ❑ 9 Flag
- ❑ 10 Infinite Loops
- ❑ 11 Initialize
- ❑ 12 Input()
- ❑ 13 Indefinite iteration
- ❑ 14 Iteration
- ❑ 15 Loop Header
- ❑ 16 Loop Body
- ❑ 17 Modulo Operator
- ❑ 18 Numerical Input
- ❑ 19 Pass Statement
- ❑ 20 Range()
- ❑ 21 Skip
- ❑ 22 Sentinel Value
- ❑ 23 Task Manager
- ❑ 24 Terminal
- ❑ 25 Update
- ❑ 26 Validate
- ❑ 27 While
- ❑ 28 While True
- ❑ 29 While else

Quiz 7

- Quiz 7A has 10 questions in 15 minutes, 10 pts
  - *10 multiple choice/true or false questions, 1 pt. for each question*
  - *Quiz 7A has two attempt, the higher grade will be selected*
  - *Submit Quiz 7A (at least 1-minute) **before** the due time to Blackboard*

- Quiz 7B has 2 code questions, 15 pts
  - *Write the Python code based on the given question*
  - *Each question will be given during the last 10-minute of each session of week 7*
  - *Quiz 7B-1 on session A, and Quiz 7B-2 on session B*
  - *Quiz 7B has one attempt*

DB 7

- Instruction:
  1) **Modify** any **three** questions from PE7_1 to PE7_5 to whatever you want for practice. Make sure to **comment** on your **modifications** (1.2 pt).
  2) What is the difference between = and ==? Give an **example** of each and **explain** your statements (0.3 pt).
  3) Submit your posts before the due date. Let's learn from each other.