**PW SKILLS**

**FUNCTIONS ASSIGNMENT**

1. \*\*Creating a Function in Python\*\*: The `def` keyword is used to create a function in Python. Below is a function to return a list of odd numbers in the range of 1 to 25.

```python

def odd\_numbers():

return [num for num in range(1, 26) if num % 2 != 0]

print(odd\_numbers())

```

2. \*\*`\*args` and `\*\*kwargs`\*\*: `\*args` and `\*\*kwargs` are used in Python functions to handle variable-length arguments. `\*args` allows you to pass a variable number of positional arguments, and `\*\*kwargs` allows you to pass a variable number of keyword arguments. Here are examples:

```python

def func\_with\_args(\*args):

for arg in args:

print(arg)

def func\_with\_kwargs(\*\*kwargs):

for key, value in kwargs.items():

print(key, ":", value)

func\_with\_args(1, 2, 3)

func\_with\_kwargs(a=1, b=2, c=3)

```

3. \*\*Iterators in Python\*\*: An iterator is an object that represents a stream of data. The `iter()` function is used to initialize an iterator object, and the `next()` method is used for iteration. Here's how to print the first five elements of the given list:

```python

my\_list = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

my\_iterator = iter(my\_list)

for \_ in range(5):

print(next(my\_iterator))

```

4. \*\*Generator Functions and `yield`\*\*: A generator function in Python is a function that returns an iterator by using `yield` instead of `return`. `yield` allows the function to yield values one at a time, suspending its state between each yield. Here's an example:

```python

def countdown(n):

while n > 0:

yield n

n -= 1

for num in countdown(5):

print(num)

```

5. \*\*Generator Function for Prime Numbers\*\*: Here's a generator function for prime numbers less than 1000 and printing the first 20 prime numbers using the `next()` method:

```python

def is\_prime(n):

if n <= 1:

return False

for i in range(2, int(n \*\* 0.5) + 1):

if n % i == 0:

return False

return True

def prime\_generator():

num = 2

count = 0

while count < 20:

if is\_prime(num):

yield num

count += 1

num += 1

prime\_gen = prime\_generator()

for \_ in range(20):

print(next(prime\_gen))

```