DOING SOMETHING USEFUL WITH PYTHON

ES 112

Brief Recap: Representing Data in Python

- Strings
- I/O and Formatting I/O
- Types
- Conditionals

Menu for Today!
Doing Something with the Data

- (A bit more on) Conditionals
- Iterations (`while`, `for` and `break`)
- Iteration patterns

Dealing with Compound Conditions

```
print("Please input X:")
x = int(input())
print("Please input Y:")
y = int(input())
print("Please input Z:")
z = int(input())


if x > y and x > z:
    print("The maximum number is", x)
elif y > x and y > z:
    print("The maximum number is", y)
else:
    print("The maximum number is", z)
```

How would you compute max of 4 numbers? 5?

A Scalable Solution for Computing Max:

```
print("Please input X:")
x = int(input())
print("Please input Y:")
y = int(input())
print("Please input Z:")
z = int(input())

max = x
if y > max :
   max = y
if z > max:
   max = z
print("The maximum number is", max)
```

Accumulator Pattern

This is elegant!
How will you extend to 4 numbers? 5? 20?
An arbitrary number of numbers?

Max of a sequence of numbers

- User gives us a sequence of positive integers
  - *The end of the sequence is marked by a negative integer*

Sentinel Value

- We need repeat this fragment of code for each number

```
if number > max :
    max = number
```

- We need to keep doing this while we haven’t seen a negative number. That is

```
number >= 0
```

The While Construct in Python

```
while condition:
    expression1
    expression2
expressionN
```

Compute the Max of a Sequence of Numbers

```
max = None
number = input(“Give me a number”)
while number >= 0:
    if(max == None or number > max):
        max = number
    number = input(“Give me a number”)
if (max == None)
    print(“You did not input any numbers”)
else:
    print("The maximum number is", max)
```

Powers of Two

```
x = 1
while (x < 1000):
    print(x)
    x *= 2
```

Multiplication Tables

```
number = 8
i = 1
while i < 11:
    print(f'{number:2d} {i:2d}s are {number*i}')
    i += 1
```

This is somewhat clumsy!! Instead, we can write

```
number = 8
for i in range(1, 11):
    print(f'{number:2d} {i:2d}s are {number*i}')
```

For loops

```
for i in range(n,m,s):
    expression1
    expression2
expressionN
```

`i in range(n,m,s)`

- `i` first gets the value n
- At the end of the loop, `i` is incremented by s

`n, n + s, n + 2s, ….`

- Continue while `i < m`
- Note `i < m` and not `i` <= m

For loops

```
for i in range(n,m,s):
    expression1
    expression2
expressionN
```

What is `range()`

`range(n,m,s)`

- `generates a list of of values [n, n+s, n+2s,…,n+ks]`
- Continue while `n+ks < m`
- Note `n+ks < m` and not `n+ks <= m`

- You can use range in 3 forms
- `range(m):` `[0, 1, …, m-1]`
- `range(n,m)` `[n, n+1, …, m - 1]`
- `range(n,m,s):` `[n, n+s, n+2s,…,n+ks]`

Back to Multiplication Tables

- How would you print the tables upto to 12 instead of 10? How do we make this kind of change easy to do?

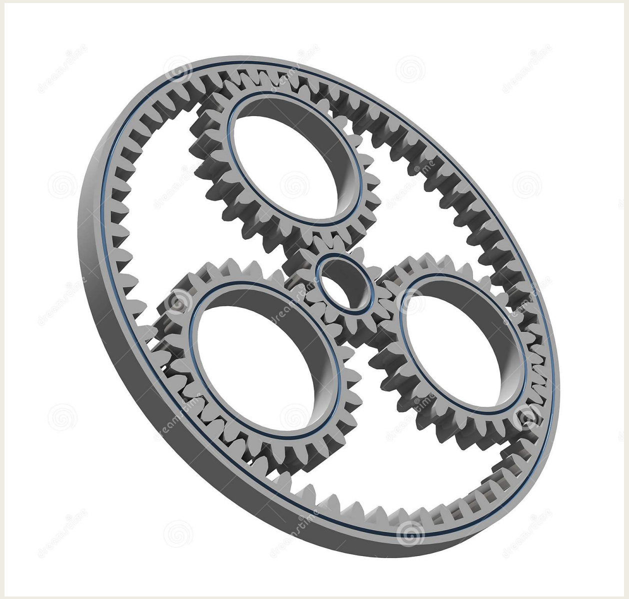
```
number = 8
sizeOfTable = 12
for i in range(1, sizeOfTable + 1):
    print(f'{number:2d} {i:2d}s are {number*i}')
```

- Issues with alignment!!! How do we make the width also changeable

```
width = 3
number = 8
sizeOfTable = 12
for i in range(1, sizeOfTable + 1):
    print(f'{number:2d} {i:2d}s are {number*i:width}')
```

Wheels within wheels: Nested Loops

- Write a program that prints all primes up to a given number number
  - *for each number* `value < number`
  - *for each* `number factor < value`
  - *check if* `factor` *is a factor of* `value`
- Note: we have a loop within a loop



Computing Primes

```
number = int(input('Give me a number '))
for value in range(2,number):
    isPrime = True
    factor = 2
    while (factor < value and isPrime):
        if (value % factor == 0):
            isPrime = False
        factor += 1
    if isPrime:
        print('%d is a prime number' % i)
```

Exiting a Loop : break statement

- the use of `isPrime` to terminate the inner loop is kludgy
- We simply want to terminate the inner loop when we find a factor
- `break` allows us to terminate the loop
- Only the inner most loop is terminated

```
while condition_1:
    while condition_2:
        statement_a
        if condition_3:   # this is typical but not essential
            break
        statement_b
    statement_c
statement_d
```

Computing Primes with a break!

```
number = int(input('Give me a number '))
for value in range(2,number):
    isPrime = True
    for factor in range(2, value)
        if (value % factor == 0):
            isPrime = False
            break
    if isPrime:
        print('%d is a prime number’ % value)
```

Infinite Loops

- Statements in the “body” of a `while` loop should make the `condition` eventually `False`
  - *What happens if this never happens?*
  - *Infinite loop*
- Infinite loops could be used to monitor the environment
- In most cases, Infinite loops are usually errors
  - *Debug by putting print statements in the body*

Comparing while and for

`while` loop

- unknown number of iterations
- End early with `break`
- Can use counter
  - *explicit initialization and incrementing necessary*
- Not always possible to express a `while` loop as a `for` loop
- Could be slightly slower

`for` loop

- known (bounded) number of iterations
- End early with `break`
- Can use counter
  - *automatic initialization and incrementing*
- A `for` loop can be written as a `while` loop
- Could be slightly faster