Week 2

Incremental Development

Arithmetic Revisited

For loops

Accumulator Pattern

String operators

Incremental Development

Makes programming much easier

Idea: Complete program in steps. Make sure current step works before moving on to the next step

Incremental development example: Adding two numbers

- Step 1: Implement user input and test
- Step 2: Add both numbers and test
- Step 3: Put everything together and clean up

Incremental Development

Makes programming much easier

Idea: Complete program in steps. Make sure current step wo
to the next step

Incremental development example: Adding two numbers

Step 1: Implement user input and test

Step 2: Add both numbers and test

Step 3: Put everything together and clean up

Has several steps!

- Call input() to prompt user
- Convert from string to integer
- Save the result in a variable

Incremental Development

Makes programming much easier

Idea: Complete program in steps. Make sure current step wor
to the next step

Incremental development example: Adding two numbers

Step 1: Implement user input and test

Step 2: Add both numbers and test

Step 3: Put everything together and clean up

Two steps:

- Add first and second number
- Save result in a variable

Incremental Development

Makes programming much easier

Idea: Complete program in steps. Make sure current step wor
to the next step

Incremental development example: Adding two numbers

Step 1: Implement user input and test

Step 2: Add both numbers and test

Step 3: Put everything together and clean up

- Remove print statements for intermediate values
- Comment code
- Output final results with desired formatting

Arithmetic Revisited

Type conversions can can happen automatically: int & float → float

Other useful functions: abs(), min(), max(), pow(), ** (exponentiate), % (modulo)

Operator precedence: same as math, left to right, mult/div before add/sub

```
4 * (2 + 3) != 4*2 + 3
```

When in doubt, use parentheses!

More powerful math functions are in the **math module** (e.g. using import math)

% Modulo

Returns the remainder of integer division

ex. 6 % 2 = 0 because 6/2 = 3 with remainder 0

ex. 6 % 4 = 2 because 6/4 = 1 with remainder 2

Question: How can we use % to determine is a number is even or odd?

Compute the tax

Compute the cost of an item after tax

- The user should input the base price and the tax
- The program should output the total cost

```
almond[w01]$ python3 tax.py

Enter the cost: 10

Enter the tax: 0.3

The total is 13.0
```

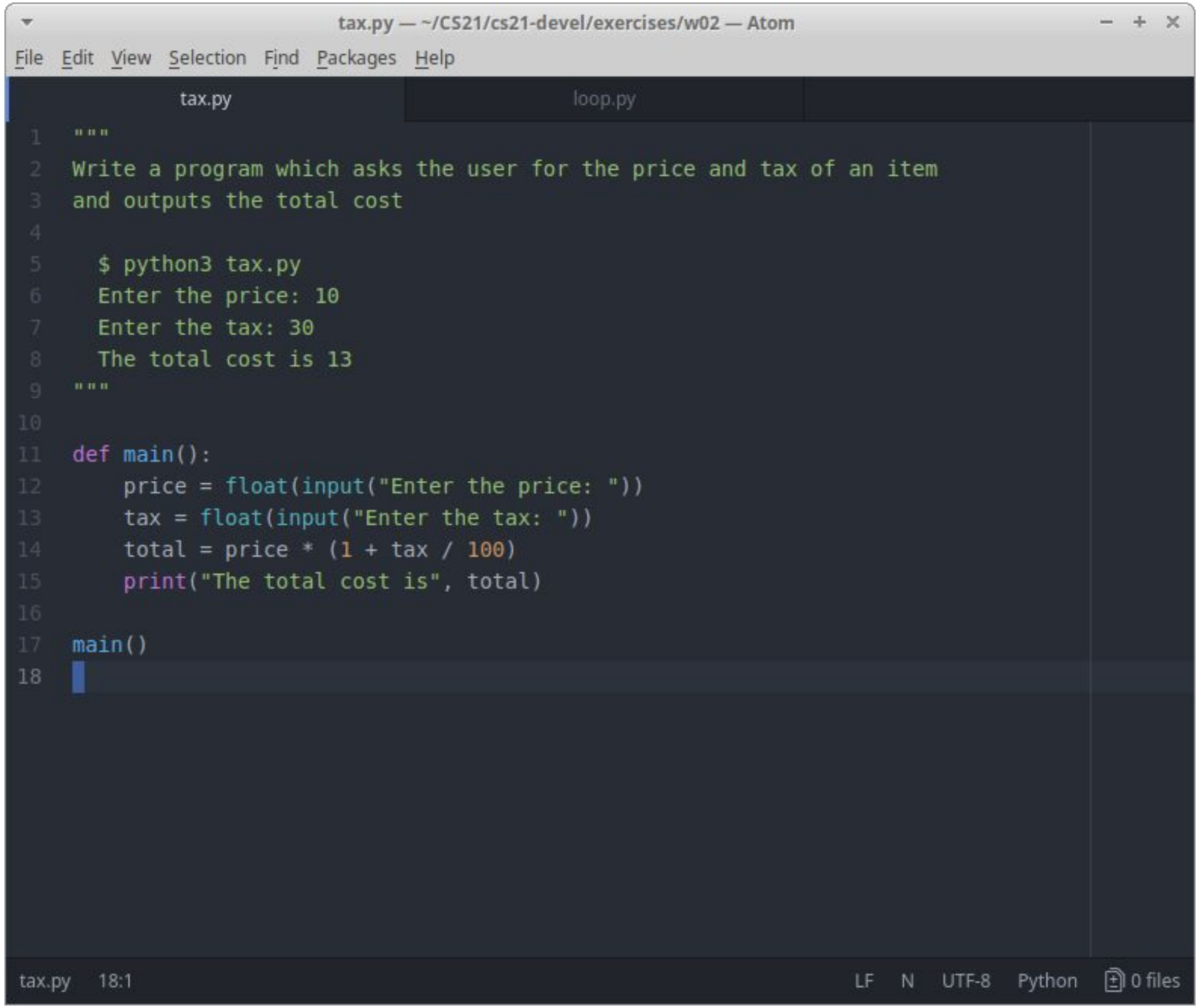
Compute the tax

Use an incremental development approach

1. Specify the algorithm on paper (4 steps)
    a. What types do we need?
    b. List some test cases (e.g. how can we tell that the program is working)?

2. Write your program in steps
    a. Implement your input and check it
    b. Implement your computation and check it
    c. Put it all together and cleanup

```
almond[w01]$ python3 tax.py

Enter the cost: 10

Enter the tax: 0.3

The total is 13.0
```


tax.py — ~/CS21/cs21-devel/exercises/w02 — Atom
File Edit View Selection Find Packages Help
tax.py
loop.py
tax.py 18:1
LF N UTF-8 Python 0 files

```python
"""
Write a program which asks the user for the price and tax of an item
and outputs the total cost

  $ python3 tax.py
  Enter the price: 10
  Enter the tax: 30
  The total cost is 13
"""

def main():
    price = float(input("Enter the price: "))
    tax = float(input("Enter the tax: "))
    total = price * (1 + tax / 100)
    print("The total cost is", total)

main()

```

Loops

Idea: repeat a set of instructions

Two kinds: **for** loops and **while** loops. Let’s talk about **for** loops first!

Real life examples

For 30 seconds, stir the pot

For each name on the list, read it aloud

In code:

```
for i in range(5):
        print(i)
```

```
for i in [1,2,3,4,5]:
        print(i)
```

Loops - Syntax

[NOTE: Syntax means the rules of a language]

```
for <var> in <sequence>:

    <body>
```

colon is important!

Indent is important!!

<var>: stores the current iteration, e.g. “which step”
<sequence>: set of values that <var> takes on
<body>: repeated for every element in <sequence>

Loops - Example

```
for i in [1,2,3,4,5]:
    print(i)
```

1. *i* is initialized to first item in the sequence. In this example, *i = 1*
2. We execute the body. In this example, we *print(i)*. Because *i = 1,* the number 1 is output to the console.
3. We then update *i* to the next item in the sequence
    a. If there are no more items, we exit the loop
    b. Otherwise, we execute the body with the current value of *i*

Loops - Specifying sequences

Two ways to specify a sequence:

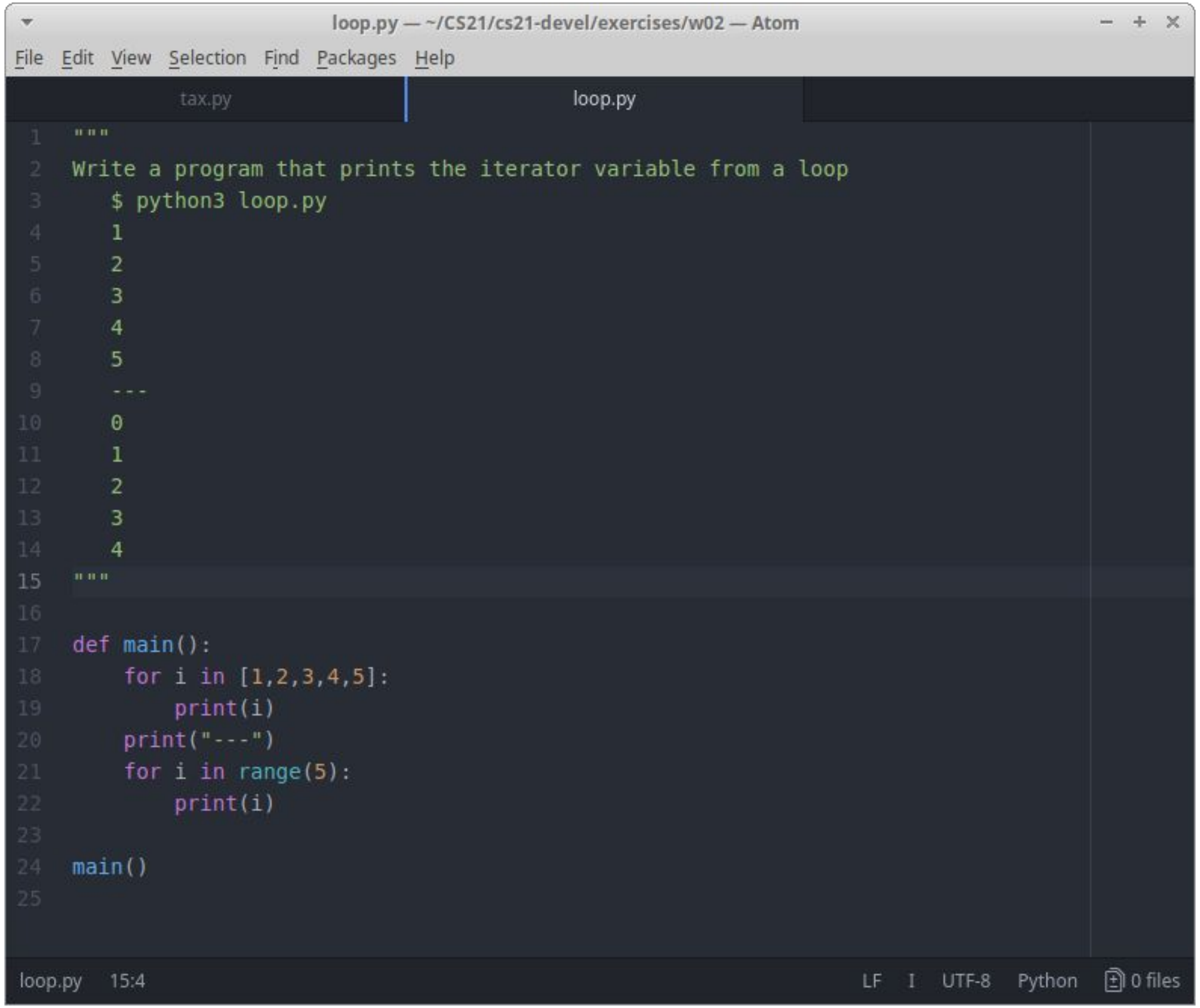
[1,2,3,4,5] ← List, Syntax uses brackets [] and commas to separate elements

range(5) ← Function which returns a list having 5 elements, namely [0,1,2,3,4]

NOTE: you'll notice that range uses 0-based indexing! If this seems weird, you're not wrong. The reason we start counting at 0 is related to how memory is laid out. It will become more clear (and comfortable) over time!

```
for i in [1,2,3,4,5]:
    print(i)
```

```
for i in range(5):
    print(i)
```


loop.py — ~/CS21/cs21-devel/exercises/w02 — Atom
File Edit View Selection Find Packages Help
tax.py
loop.py
loop.py 15:4
LF UTF-8 Python 0 files

```python
"""
Write a program that prints the iterator variable from a loop
   $ python3 loop.py
   1
   2
   3
   4
   5
   ---
   0
   1
   2
   3
   4
"""

def main():
    for i in [1,2,3,4,5]:
        print(i)
    print("---")
    for i in range(5):
        print(i)

main()
```

Accumulator pattern

Idea: have a variable which accumulates a value over multiple iterations

Examples: a bank balance, counting cookies

Algorithm

1. Initialize an accumulator variable
2. For each item in the sequence
    a. Update the accumulator
3. Use the accumulator

Example - Compute the total

Remember the algorithm to compute the total we talked about the first day?

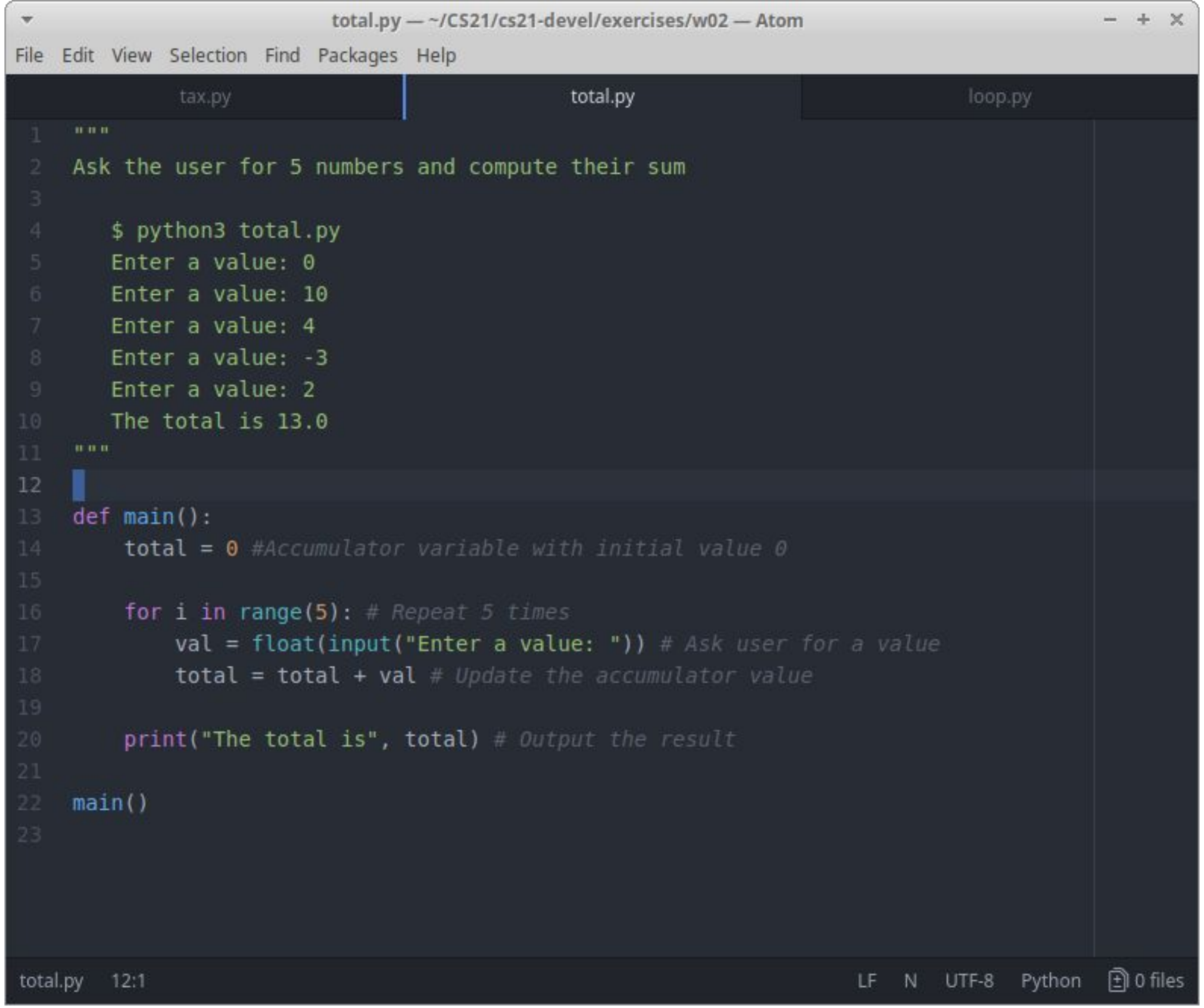
```
The total starts at zero
Repeat the following steps 5 times:
    Listen for the next number
    Update total using the new number
Say what the total was
```

Example - Compute the total

Let’s implement it using an **accumulator pattern** and **incremental development!**

```
The total starts at zero
Repeat the following steps 5 times:
    Listen for the next number
    Update total using the new number
Say what the total was
```

- What are we accumulating? total
- What does the accumulator equal to start? 0
- How to update the accumulator each loop iteration? total = total + new_value
- How many times should we loop? 5
- How to get the input? Ask User
- How do we know if our accumulator is correct? Check total with known values


total.py — ~/CS21/cs21-devel/exercises/w02 — Atom
File Edit View Selection Find Packages Help
tax.py
total.py
loop.py

```python
"""
Ask the user for 5 numbers and compute their sum

   $ python3 total.py
   Enter a value: 0
   Enter a value: 10
   Enter a value: 4
   Enter a value: -3
   Enter a value: 2
   The total is 13.0
"""

def main():
    total = 0 #Accumulator variable with initial value 0

    for i in range(5): # Repeat 5 times
        val = float(input("Enter a value: ")) # Ask user for a value
        total = total + val # Update the accumulator value

    print("The total is", total) # Output the result

main()

```

Aside: Shorthand assignments

Idea: Because accumulators are so common in programming, most languages define the following operators (referred to as *syntactic sugar)*

cost += 10 ← the same as **cost = cost + 10**

cost -= 10 ← the same as **cost = cost - 10**

cost *= 10 ← the same as **cost = cost * 10**

cost /= 10 ← the same as **cost = cost / 10**

Exercise - Implement factorial

The factorial operator **n!** is defined as follows for positive integers

n! = n * (n-1) * (n-2) * … (2) * (1)

where one and zero are special cases:

1! = 1 and 0! = 1

Examples:

8! = 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1 = 40320

0! = 1

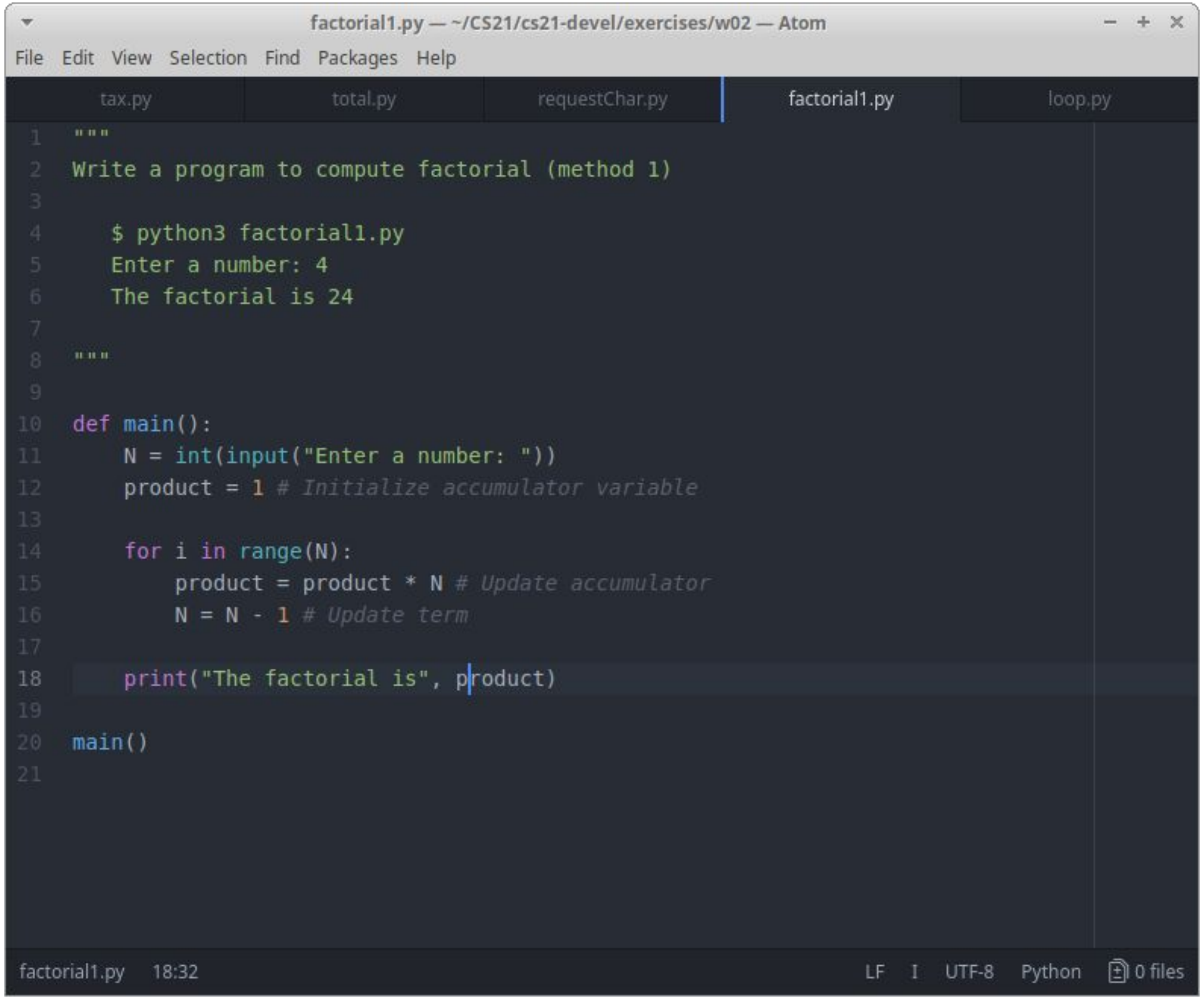
- What are we accumulating?
- What does the accumulator equal to start?
- How to update the accumulator each loop iteration?
- How many times should we loop?
- How to get the input?
- How do we know if our accumulator is correct?

Factorial accumulator

- What are we accumulating? product
- What does the accumulator equal to start? 1
- How to update the accumulator each loop iteration? product = product * nextValue
- How many times should we loop? N times
- How to get the input? Ask the user for N
- How do we know if our accumulator is correct? Check example values work

Factorial algorithm

```
Ask the user for a number N
Compute the factorial
    Initialize accumulator variable
    Repeat N times
        Multiply accumulator by next value
        (For debugging) Output the accumulator
Output the result
```

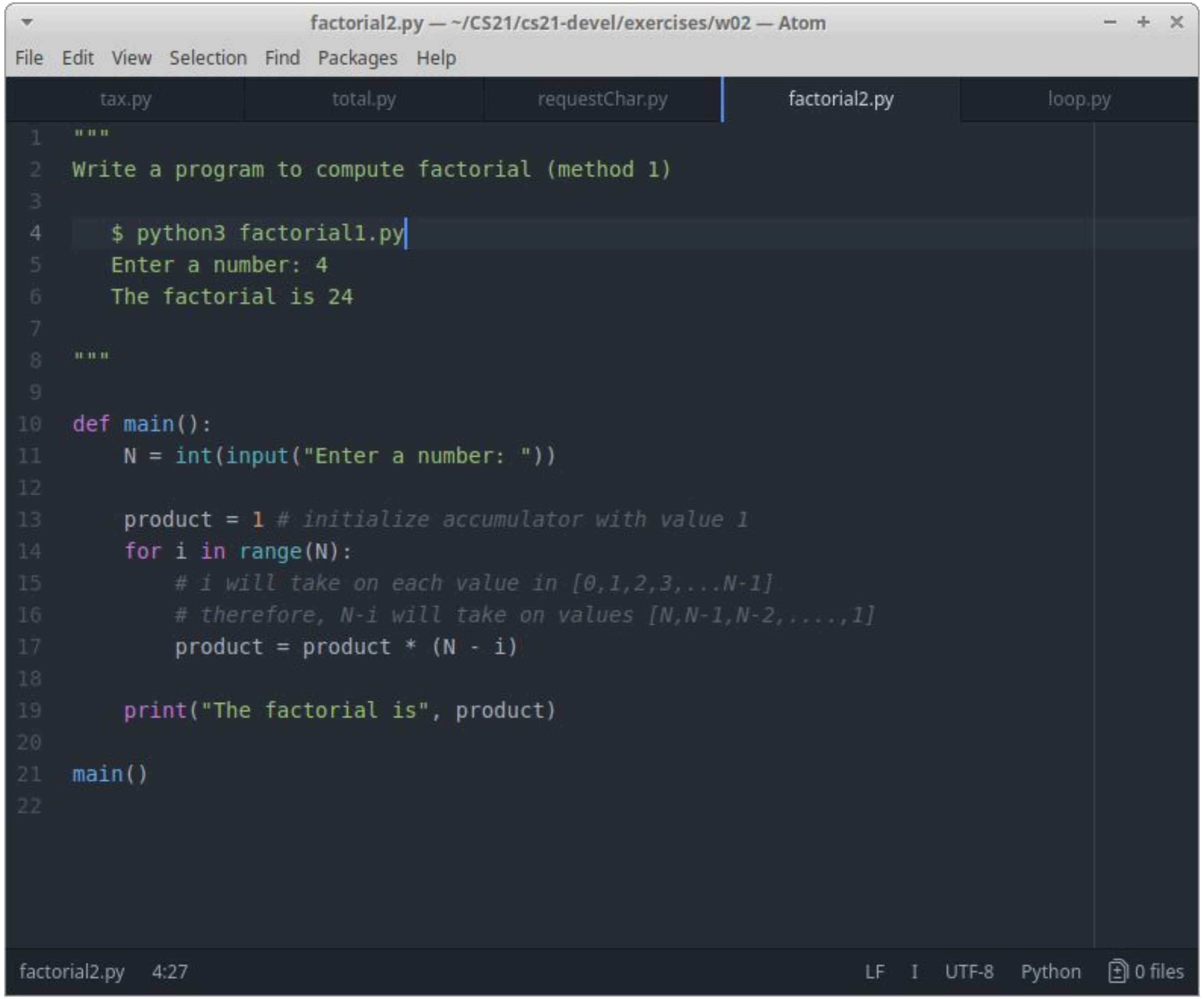

factorial1.py — ~/CS21/cs21-devel/exercises/w02 — Atom
File Edit View Selection Find Packages Help
tax.py total.py requestChar.py factorial1.py loop.py
factorial1.py 18:32 LF UTF-8 Python 0 files

```python
"""
Write a program to compute factorial (method 1)

   $ python3 factorial1.py
   Enter a number: 4
   The factorial is 24

"""

def main():
    N = int(input("Enter a number: "))
    product = 1 # Initialize accumulator variable

    for i in range(N):
        product = product * N # Update accumulator
        N = N - 1 # Update term

    print("The factorial is", product)

main()
```

How do values change as we iterate over the loop?

N = 3
product = 1

Iteration	**i**	**N**	**product = product * N**
1	0	3	product = 1 * 3 = 3
2	1	2	product = 3 * 2 = 6
3	2	1	product = 6 * 1 = 6

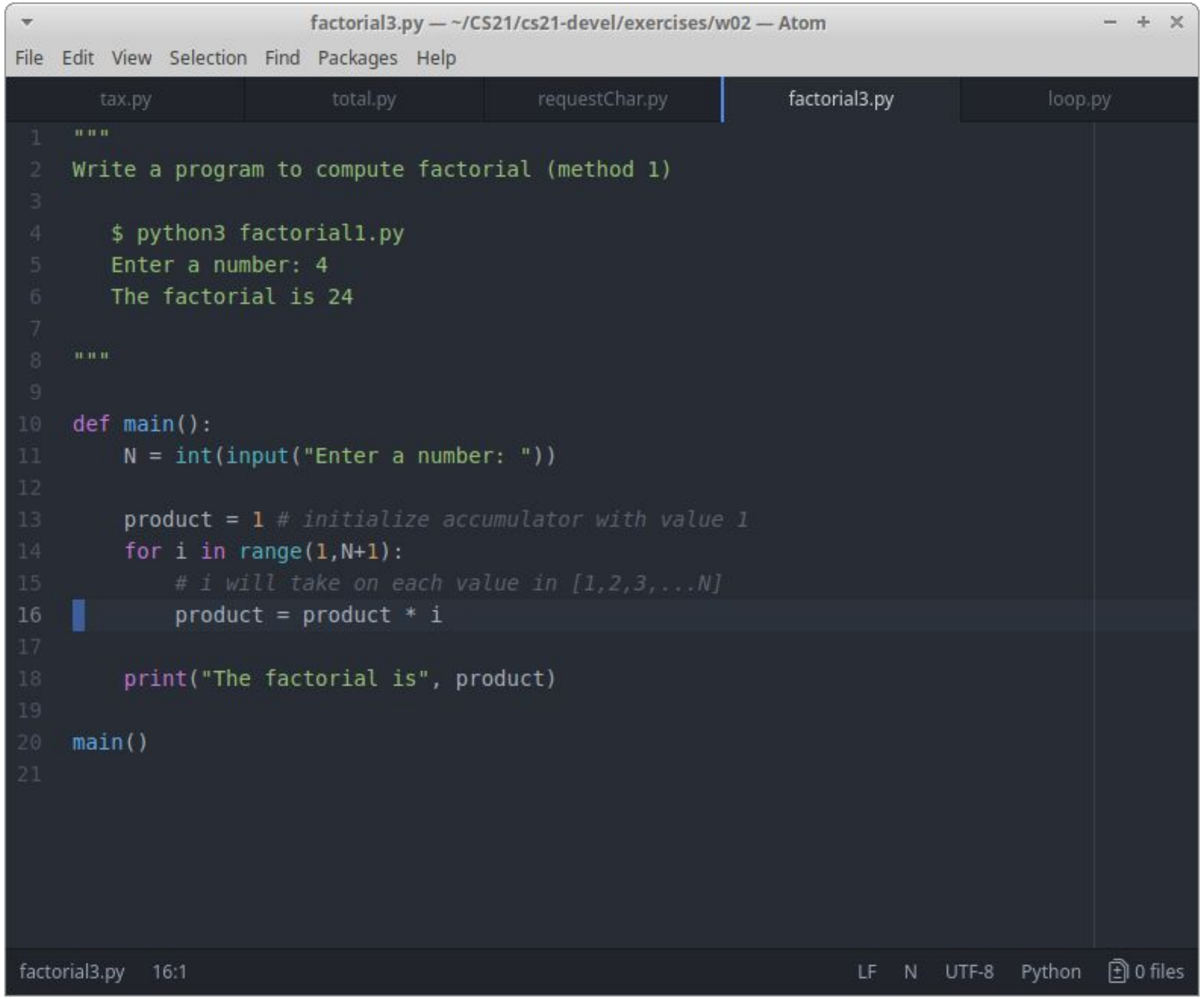

factorial2.py — ~/CS21/cs21-devel/exercises/w02 — Atom
File Edit View Selection Find Packages Help
tax.py total.py requestChar.py factorial2.py loop.py
factorial2.py 4:27 LF UTF-8 Python 0 files

```python
"""
Write a program to compute factorial (method 1)

   $ python3 factorial1.py
   Enter a number: 4
   The factorial is 24

"""

def main():
    N = int(input("Enter a number: "))

    product = 1 # initialize accumulator with value 1
    for i in range(N):
        # i will take on each value in [0,1,2,3,...N-1]
        # therefore, N-i will take on values [N,N-1,N-2,....,1]
        product = product * (N - i)

    print("The factorial is", product)

main()
```

How do values change as we iterate over the loop?

N = 3
product = 1

Iteration	**i**	**N**	**product = product * (N-i)**
1	0	3	product = 1 * (3 - 0) = 3
2	1	3	product = 3 * (3 - 1) = 6
3	2	3	product = 6 * (3 - 2) = 6


factorial3.py — ~/CS21/cs21-devel/exercises/w02 — Atom
File Edit View Selection Find Packages Help
tax.py
total.py
requestChar.py
factorial3.py
loop.py
factorial3.py 16:1
LF N UTF-8 Python 0 files

```python
"""
Write a program to compute factorial (method 1)

   $ python3 factorial1.py
   Enter a number: 4
   The factorial is 24

"""

def main():
    N = int(input("Enter a number: "))

    product = 1 # initialize accumulator with value 1
    for i in range(1,N+1):
        # i will take on each value in [1,2,3,...N]
        product = product * i

    print("The factorial is", product)

main()
```

How do values change as we iterate over the loop?

N = 3
product = 1

Iteration	**i**	**product = product * i**
1	1	product = 1 * 1 = 1
2	2	product = 1 * 2 = 2
3	3	product = 2 * 3 = 6

Strings revisited

Strings support operators, just like numbers do

> * represents repetition
>
> \+ represents concatenation
>
> len(text) returns the number of characters in a string
>
> [] index operators, which use 0-based indexing
>
> “” is an empty string

Strings are **immutable**, which means that you can’t change the value of a string after it’s created

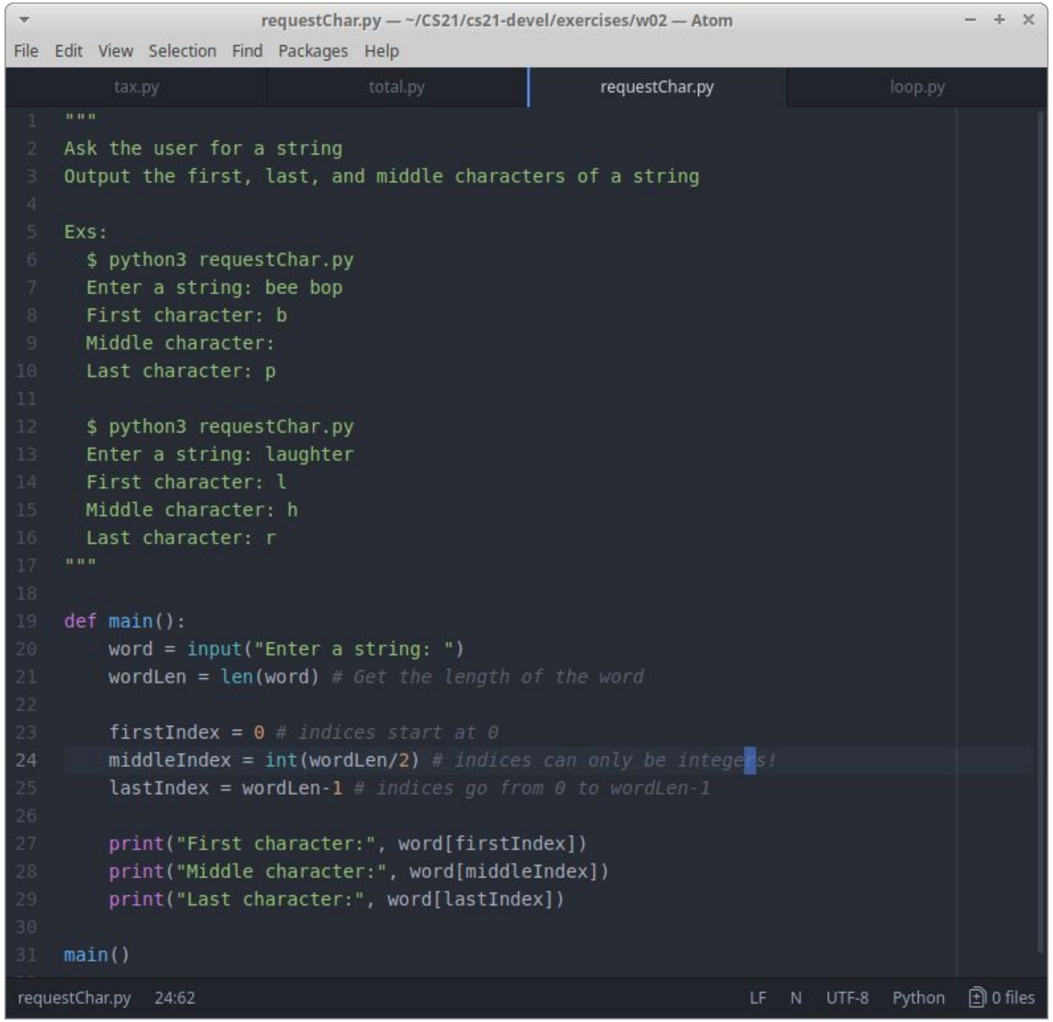
Exercise - Output the requested character

Ask the user for a string

Output the first character, the last character, and the middle character

Hint: Use indexing

```
$ python3 requestChar.py
Enter a word: laughter
first:  l
middle:  h
last:  r

$ python3 requestChar.py
Enter a word: bee bop
first:  b
middle:
last:  p
```


requestChar.py — ~/CS21/cs21-devel/exercises/w02 — Atom
File Edit View Selection Find Packages Help
tax.py
total.py
requestChar.py
loop.py
requestChar.py 24:62
LF N UTF-8 Python 0 files

```python
"""
Ask the user for a string
Output the first, last, and middle characters of a string

Exs:
  $ python3 requestChar.py
  Enter a string: bee bop
  First character: b
  Middle character:
  Last character: p

  $ python3 requestChar.py
  Enter a string: laughter
  First character: l
  Middle character: h
  Last character: r
"""

def main():
    word = input("Enter a string: ")
    wordLen = len(word) # Get the length of the word

    firstIndex = 0 # indices start at 0
    middleIndex = int(wordLen/2) # indices can only be integers!
    lastIndex = wordLen-1 # indices go from 0 to wordLen-1

    print("First character:", word[firstIndex])
    print("Middle character:", word[middleIndex])
    print("Last character:", word[lastIndex])

main()
```