```
(base) PS C:\Users\Abi Rahman> python

Python 3.10.9 | packaged by conda-forge | (main, Jan 11 2023, 15:15:40) [MSC v.1916 64 bit
(AMD64)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>> x = int(input("Please enter an integer: "))

Please enter an integer: 12

>>> if x < 0:

...    x = 0

...    print('Negaitve change to zero')

... elif x == 0:

...    print('Zero')

... elif x == 1:

...    print('Single')

... else:

...    print('More')

...

More

>>> # ^^^if statement^^^

>>>

>>> # for statement

>>> words = ['cat', 'window', 'defenestrate']

>>> for w in words:

...     print(w, len(w))

...

cat 3

window 6

defenestrate 12

>>>

>>> users = {'Hans': 'active', 'Éléonore': 'inactive', '景太郎': 'active'}

>>> # Strategy:  Iterate over a copy

>>> for user, status in users.copy().items():
```

```
...     if status == 'inactive':
...         del users[user]
...
>>> # Strategy:  Create a new collection
>>> active_users = {}
>>> for user, status in users.items():
...     if status == 'active':
...         active_users[user] = status
...
>>> # the range() function
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
>>> list(range(5, 10))
[5, 6, 7, 8, 9]
>>> list(range(0, 10, 3))
[0, 3, 6, 9]
>>> list(range(-10, -100, -30))
[-10, -40, -70]
>>>
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print(i, a[i])
...
0 Mary
1 had
```

```
2 a
3 little
4 lamb
>>>
>>> range(10)
range(0, 10)
>>> sum(range(4))  # 0 + 1 + 2 + 3
6
>>>
>>> # break and continue Statements, and else Clauses on Loops
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     else:
...         # loop fell through without finding a factor
...         print(n, 'is a prime number')
...
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
>>>
>>>
>>> for num in range(2, 10):
...     if num % 2 == 0:
```

```
...         print("Found an even number", num)
...         continue
...     print("Found an odd number", num)
...
Found an even number 2
Found an odd number 3
Found an even number 4
Found an odd number 5
Found an even number 6
Found an odd number 7
Found an even number 8
Found an odd number 9
>>>
>>> # pass Statements
>>> while True:
...     pass
...
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyboardInterrupt
>>>
>>>
>>>
>>> class MyEmptyClass:
...     pass
...
>>> def initlog(*args):
...     pass
...
>>>
>>> from enum import Enum
>>> class Color(Enum):
...     RED = 'red'
...     GREEN = 'green'
...     BLUE = 'blue'
...
>>> color = Color(input("Enter your choice of 'red', 'blue' or 'green': "))
Enter your choice of 'red', 'blue' or 'green': blue
>>>
>>> match color:
...     case Color.RED:
...         print("I see red!")
...     case Color.GREEN:
...         print("Grass is green")
...     case Color.BLUE:
...         print("I'm feeling the blues :(")
...
I'm feeling the blues :(
>>>
>>> # Defining Functions
>>> def fib(n):    # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print(a, end=' ')
...         a, b = b, a+b
...     print()
...
>>> fib
<function fib at 0x000001F03D217910>
>>> f = fib
>>> f(100)
```

```
0 1 1 2 3 5 8 13 21 34 55 89
>>> fib(0)

>>> print(fib(0))

None
>>>
>>> def fib2(n):  # return Fibonacci series up to n
...     """Return a list containing the Fibonacci series up to n."""
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a)    # see below
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100)    # call it
>>> f100
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>>
>>>
>>> # More on Defining Functions
>>> #  1. Default Argument Values
>>> def ask_ok(prompt, retries=4, reminder='Please try again!'):
...     while True:
...         ok = input(prompt)
...         if ok in ('y', 'ye', 'yes'):
...             return True
...         if ok in ('n', 'no', 'nop', 'nope'):
...             return False
...         retries = retries - 1
...         if retries < 0:
...             raise ValueError('invalid user response')
...         print(reminder)
...
>>> i = 5
>>> def f(arg=i):
...     print(arg)
...
>>> i = 6
>>> f()
5
>>>
>>> def f(a, L=[]):
...     L.append(a)
...     return L
...
>>> print(f(1))
[1]
>>> print(f(2))
[1, 2]
>>> print(f(3))
[1, 2, 3]
>>>
>>> def f(a, L=None):
...     if L is None:
...         L = []
...     L.append(a)
...     return L
...
>>>
>>> # 2. Keyword Arguments
>>> def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
...     print("-- This parrot wouldn't", action, end=' ')
...     print("if you put", voltage, "volts through it.")
...     print("-- Lovely plumage, the", type)
...     print("-- It's", state, "!")
...
>>> parrot(1000)                                  # 1 positional argument
-- This parrot wouldn't voom if you put 1000 volts through it.
-- Lovely plumage, the Norwegian Blue
-- It's a stiff !
>>> parrot(voltage=1000)                          # 1 keyword argument
-- This parrot wouldn't voom if you put 1000 volts through it.
-- Lovely plumage, the Norwegian Blue
-- It's a stiff !
>>> parrot(voltage=1000000, action='VOOOOOM')             # 2 keyword arguments
-- This parrot wouldn't VOOOOOM if you put 1000000 volts through it.
-- Lovely plumage, the Norwegian Blue
-- It's a stiff !
>>> parrot(action='VOOOOOM', voltage=1000000)             # 2 keyword arguments
-- This parrot wouldn't VOOOOOM if you put 1000000 volts through it.
-- Lovely plumage, the Norwegian Blue
-- It's a stiff !
>>> parrot('a million', 'bereft of life', 'jump')         # 3 positional arguments
-- This parrot wouldn't jump if you put a million volts through it.
-- Lovely plumage, the Norwegian Blue
-- It's bereft of life !
>>> parrot('a thousand', state='pushing up the daisies')  # 1 positional, 1 keyword
-- This parrot wouldn't voom if you put a thousand volts through it.
-- Lovely plumage, the Norwegian Blue
-- It's pushing up the daisies !
>>>
```

```
>>> def cheeseshop(kind, *arguments, **keywords):
...     print("-- Do you have any", kind, "?")
...     print("-- I'm sorry, we're all out of", kind)
...     for arg in arguments:
...         print(arg)
...     print("-" * 40)
...     for kw in keywords:
...         print(kw, ":", keywords[kw])
...
>>> cheeseshop("Limburger", "It's very runny, sir.",
...            "It's really very, VERY runny, sir.",
...            shopkeeper="Michael Palin",
...            client="John Cleese",
...            sketch="Cheese Shop Sketch")
-- Do you have any Limburger ?
-- I'm sorry, we're all out of Limburger
It's very runny, sir.
It's really very, VERY runny, sir.
----------------------------------------
shopkeeper : Michael Palin
client : John Cleese
sketch : Cheese Shop Sketch
>>>
>>># 3. Special parameters
>>>def f(pos1, pos2, /, pos_or_kwd, *, kwd1, kwd2):
          -----------    ----------     ----------
        |               |              |
        |    Positional or keyword     |
        |                              - Keyword only
          -- Positional only
```

```
>>> # Function Examples
>>> def standard_arg(arg):
...     print(arg)
...
>>> def pos_only_arg(arg, /):
...     print(arg)
...
>>> def kwd_only_arg(*, arg):
...     print(arg)
...
>>> def combined_example(pos_only, /, standard, *, kwd_only):
...     print(pos_only, standard, kwd_only)
...
>>> standard_arg(2)
2
>>> standard_arg(arg=2)
2
>>> pos_only_arg(1)
1
>>> kwd_only_arg(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: kwd_only_arg() takes 0 positional arguments but 1 was given
>>> kwd_only_arg(arg=3)
3
>>>
>>> combined_example(1, 2, kwd_only=3)
1 2 3
>>> combined_example(1, standard=2, kwd_only=3)
1 2 3
>>> def foo(name, /, **kwds):
```

```
...    return 'name' in kwds
...
>>> foo(1, **{'name': 2})
True
>>>
>>> # Recap
>>> def write_multiple_items(file, separator, *args):
...    file.write(separator.join(args))
...
>>> def concat(*args, sep="/"):
...    return sep.join(args)
...
>>> concat("earth", "mars", "venus")
'earth/mars/venus'
>>> concat("earth", "mars", "venus", sep=".")
'earth.mars.venus'
>>>
>>> # 5. Unpacking Argument Lists
>>> list(range(3, 6))          # normal call with separate arguments
[3, 4, 5]
>>> args = [3, 6]
>>> list(range(*args))          # call with arguments unpacked from a list
[3, 4, 5]
>>> def parrot(voltage, state='a stiff', action='voom'):
...    print("-- This parrot wouldn't", action, end=' ')
...    print("if you put", voltage, "volts through it.", end=' ')
...    print("E's", state, "!")
...
>>> d = {"voltage": "four million", "state": "bleedin' demised", "action": "VOOM"}
>>> parrot(**d)
-- This parrot wouldn't VOOM if you put four million volts through it. E's bleedin' demised !
```

```
>>>
>>> # 6. Lambda Expressions
>>> def make_incrementor(n):
...     return lambda x: x + n
...
>>> f = make_incrementor(42)
>>> f(0)
42
>>> f(1)
43
>>> pairs = [(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')]
>>> pairs.sort(key=lambda pair: pair[1])
>>> pairs
[(4, 'four'), (1, 'one'), (3, 'three'), (2, 'two')]
>>>
>>> # 7. Documentation Strings
>>> def my_function():
...     """Do nothing, but document it.
...
...     No, really, it doesn't do anything.
...     """
...     pass
...
>>> print(my_function.__doc__)
Do nothing, but document it.

    No, really, it doesn't do anything.

>>>>>> #8. Function Annotations
>>> def f(ham: str, eggs: str = 'eggs') -> str:
...     print("Annotations:", f.__annotations__)
...     print("Arguments:", ham, eggs)
...     return ham + ' and ' + eggs
...
>>> f('spam')
Annotations: {'ham': <class 'str'>, 'eggs': <class 'str'>, 'return': <class 'str'>}
Arguments: spam eggs
'spam and eggs'
>>>
```