Python

Functions

Modular Programming

- Break a large problem into smaller pieces
  - Smaller pieces sometimes called ‘subroutines’,‘procedures’ or **functions**
  - Why?
    - Helps manage complexity
      - Smaller blocks of code
      - Easier to read
    - Encourages re-use of code
      - Within a particular program or across different programs
    - Allows independent development of code
    - Provides a layer of ‘abstraction’
      - Hides the details of complex solutions
- Python has several built-in functions

Modular Programming

- Break a large problem into smaller pieces
  - Smaller pieces sometimes called ‘subroutines’,‘procedures’ or **functions**
  - Why?
    - Helps manage complexity
      - Smaller blocks of code
      - Easier to read
    - Encourages re-use of code
      - Within a particular program or across different programs
    - Allows independent development of code
    - Provides a layer of ‘abstraction’
      - Hides the details of complex solutions
- Python has several built-in functions

```
print(“Hello world”)
```

More on print function

- One of the most common functions we've been using
- Important in our standalone Python programs for providing output
- Additional arguments to the print function include separator, end character
  - Separator: sep = ' '
  - End character: end = '\n'

```
print("09","12",sep="-",end="-2018\n")
print("alper", "uysal",sep=".",end="@")
print("alanya","edu","tr",sep=".")
```

09-12-2018

alper.uysal@alanya.edu.tr

Input function

- Reading strings from the keyboard
- variable = input('Prompt')
- You have to convert the input to the desired type.

```
>>> name = input("Name?")
Name?Ozlem
>>> age = input("Age?")
Age?37
>>> name
'Ozlem'
>>> age
'37'
```

Functions

- Functions are **named sequences of statements that perform some action**
- Functions accept **parameters** in a **parameter list** and may return values
- Functions which do not return a value are called **void functions**
- A **function call** is the activation of a function

```
a = 55.66
b = int(a)
```

Variable storing a return value (b)

Function call (int)

Parameter (a)

Type Converter Functions

- **int, float** and **str**, which will (attempt to) convert their arguments into types int, float and str respectively. We call these type converter functions.

Type Converter Functions

- The **int** function can take a floating point number or a string, and turn it into an int. For floating point numbers, it discards the decimal portion of the number — a process we call truncation towards zero on the number line.

```
>>> int(3.14)
3
>>> int(3.9999)              # This doesn't round to the closest int!
3
>>> int(3.0)
3
>>> int(-3.999)              # Note that the result is closer to zero
-3
>>> int(minutes / 60)
10
>>> int("2345")              # Parse a string to produce an int
2345
>>> int(17)                  # It even works if arg is already an int
17
>>> int("23 bottles")
```

Type Converter Functions

- The type converter **float** can turn an integer, a float, or a syntactically legal string into a float:

```
>>> float(17)
17.0
>>> float("123.45")
123.45
```

- The type converter str turns its argument into a string:

```
>>> str(17)
'17'
>>> str(123.45)
'123.45'
```

Using Modules

- **Modules** are collections of related functions stored in one file
- **Module Objects** are the namespaces under which functions stored in a module are accessed
- A module is imported with the **import** keyword
- Functions within a module are accessed using **dot notation** (ex. **`math.sin(value)`**)

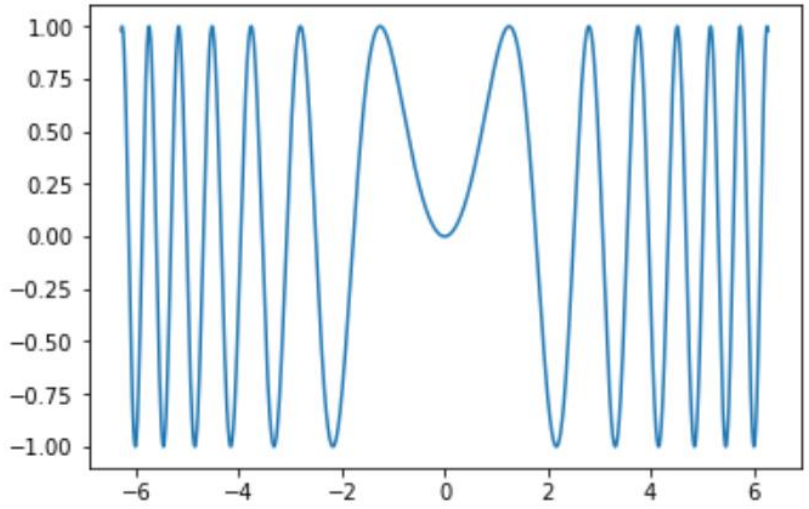
In [8]:
```
import math
a = math.sin(math.pi)
print(a)
```

```
1.2246467991473532e-16
```

In [10]:
```
import matplotlib
import matplotlib.pyplot
import numpy
from numpy import pi

x = numpy.linspace(-2*pi,2*pi,2000)
y = numpy.sin(x**2)

matplotlib.pyplot.plot(x,y)
matplotlib.pyplot.show()
```



Using Modules

- **Modules** are collections of related functions stored in one file
- **Module Objects** are the namespaces under which functions stored in a module are accessed
- A module is imported with the **import** keyword
- Functions within a module are accessed using **dot notation** (ex. `math.sin(value)`)

In [8]:
```
import math
a = math.sin(math.pi)
print(a)
```

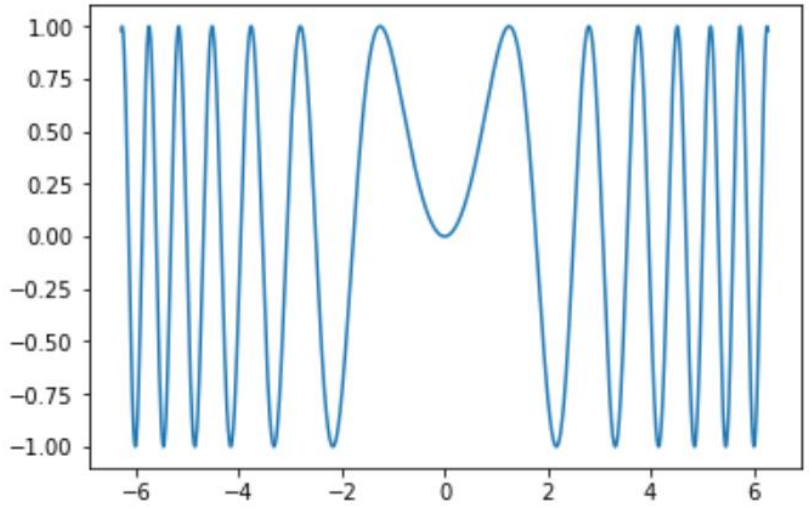
```
1.2246467991473532e-16
```

In [10]:
```
import matplotlib
import matplotlib.pyplot
import numpy
from numpy import pi

x = numpy.linspace(-2*pi,2*pi,2000)
y = numpy.sin(x**2)

matplotlib.pyplot.plot(x,y)
matplotlib.pyplot.show()
```

Module Objects



Using Modules

- **Modules** are collections of related functions stored in one file
- **Module Objects** are the namespaces under which functions stored in a module are accessed
- A module is imported with the **import** keyword
- Functions within a module are accessed using **dot notation** (ex. `math.sin(value)`)

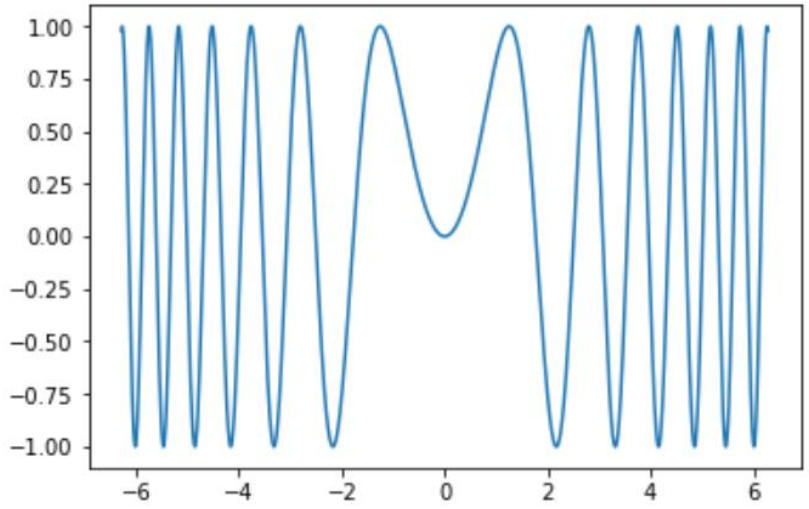
In [8]:
```
import math
a = math.sin(math.pi)
print(a)
```

```
1.2246467991473532e-16
```

In [10]:
```
import matplotlib
import matplotlib.pyplot
import numpy
from numpy import pi

x = numpy.linspace(-2*pi,2*pi,2000)
y = numpy.sin(x**2)

matplotlib.pyplot.plot(x,y)
matplotlib.pyplot.show()
```



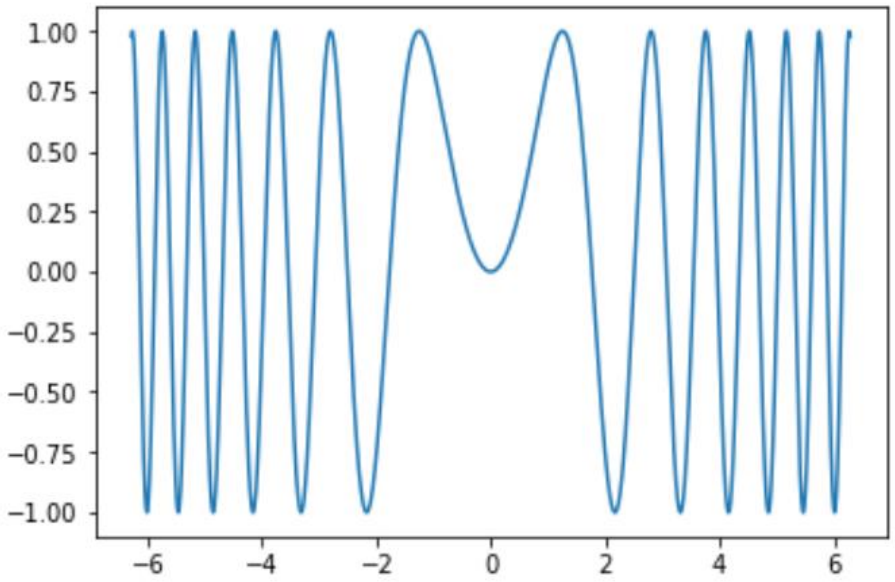
Aliases

- Shortened names to refer to module objects
- Declared using the **as** keyword

In [11]:
```
import matplotlib
import matplotlib.pyplot as pt
import numpy as np
from numpy import pi

x = np.linspace(-2*pi,2*pi,2000)
y = np.sin(x**2)

pt.plot(x,y)
pt.show()
```



Composition

- Using an expression as part of a larger statement
- Anywhere a value can be used, a function which returns a value can be used in its place

```
In [16]: import math as m
         x = 5.0 * m.pi
         print(m.sin(m.cos(100*m.tan(x))))

         0.8414709848078965
```

- In this example, a call to math.tan is a parameter for a call to math.cos which in turn is a parameter to math.sin, which is in turn a parameter to print.
  - This works because tan returns a float, which is a valid parameter value for cos, which in turn returns a float, which is a valid parameter value for sin, etc.

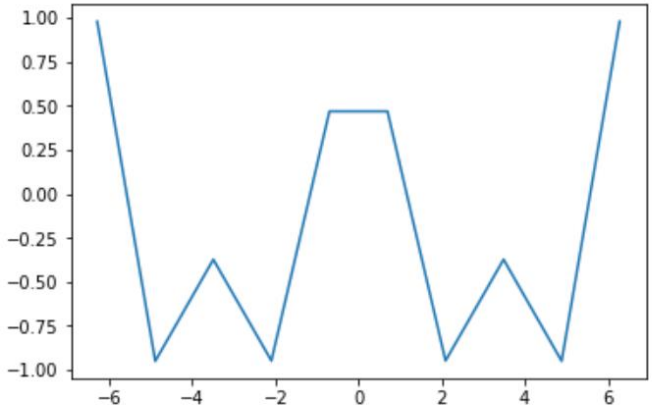
Defining Simple Functions

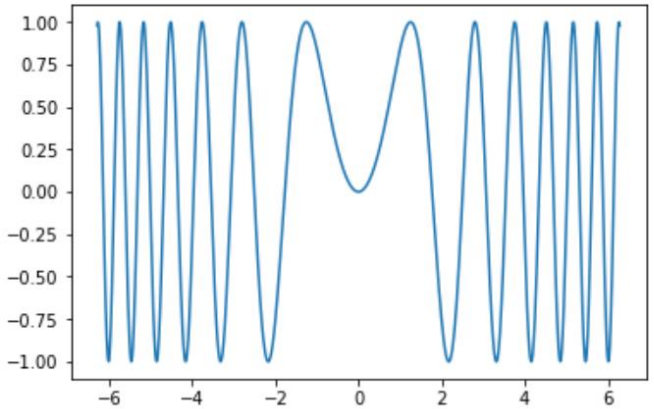
- A function is defined with the **def** keyword
- Functions have two parts, a **header** and a **body**
- The header gives the function a name and defines allowed parameters
  - The header is the first line
- The body contains the statements which define the functions action
  - Note how the body is indented
- Together, the header and body is the **function definition**

In [20]:
```
import matplotlib
import matplotlib.pyplot as pt
import numpy as np
from numpy import pi

###### Generate Plot ######
def generate_plot(min, max, steps):
    x = np.linspace(min, max, steps)
    y = np.sin(x**2)
    pt.plot(x,y)
    pt.show()

###### Main Function ######
generate_plot(-2*pi,2*pi,10)
generate_plot(-2*pi,2*pi,1000)
```





Variable Scope

- Variables which are declared within a function or are in the parameter list can only be used within that function
  - These variables are said to be **local** to that function
  - Variables declared in the main function cannot be used in a function
  - Where a variable is valid, is the **scope** of the variable

```
In [24]: def print_sum(a, b):
             s = a + b
             print(s)

         x = 5
         y = 6
         print_sum(x, y)
         print(s)
```

s is local to print_sum, it can be used here

not here

```
11

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-24-27dd9ae086d4> in <module>()
      6 y = 6
      7 print_sum(x, y)
----> 8 print(s)

NameError: name 's' is not defined
```

Note the error

Variable Scope

- Variables which are declared within a function or are in the parameter list can only be used within that function
  - These variables are said to be **local** to that function
  - Variables declared in the main function cannot be used in a function
  - Where a variable is valid, is the **scope** of the variable
- Parameters are copied in order

```
In [24]: def print_sum(a, b):
             s = a + b
             print(s)

         x = 5
         y = 6
         print_sum(x, y)
         print(s)
```

```
11

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-24-27dd9ae086d4> in <module>()
      6 y = 6
      7 print_sum(x, y)
----> 8 print(s)

NameError: name 's' is not defined
```

More Details

- Functions must be defined before they can be used
- Locally defined functions can be used within other locally defined functions
- The main function is the entry point to the program
- The main function has no header and is not indented
- The main function follows all function definitions

```
In [27]: def print_hello():
             print("hello")

         def print_hellohello():
             print_hello()
             print_hello()

         x = 5
         y = 7
         print(x + y)
         print_hellohello()
```

print_hello definition

print_hellohello definition

main function definition

```
12
hello
hello
```

Getting Help

- Functions and modules usually come with brief explanations
- To list functions in a module:
  - dir(module)
- To see all of Python’s built-in functions:
  - dir(__builtins__)
- To get help with a specific function:
  - help(function)
- Python’s general help utility:
  - help()

Resources

- Downey, A. (2016) *Think Python, Second Edition* Sebastopol, CA: O'Reilly Media
- Wentworth P., Elkner J., Downey A., and Meyers C. How to Think Like a Computer Scientist: Learning with Python 3
- Bryan Burlingame's course notes