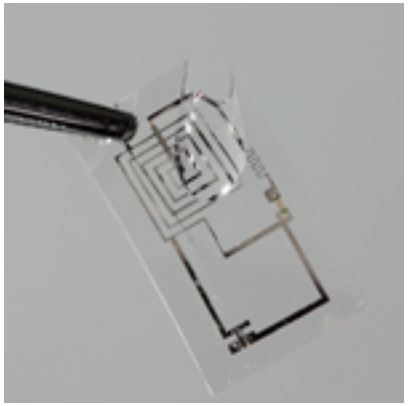
Researchers at the University of Illinois, in collaboration with Tufts University and Northwestern University, have demonstrated a new type of biodegradable electronics technology that could introduce new design paradigms for medical implants, environmental monitors and consumer devices.



Three application areas appear particularly promising. First are medical implants that perform important diagnostic or therapeutic functions for a useful amount of time and then simply dissolve and resorb in the body. Second are environmental monitors, such as wireless sensors that are dispersed after a chemical spill, that degrade over time to eliminate any ecological impact. Third are consumer electronic systems or sub-components that are compostable, to reduce electronic waste streams generated by devices that are frequently upgraded, such as cellphones or other portable devices.

http://news.illinois.edu/news/12/0927transient_electronics_JohnRogers.html

- Submitted Questions

Using Graphical Objects

- The graphics library has a better solution. Graphical objects have a clone method that will make a copy of the object
-

```
>>> # Correct way to create two circles, using
clone
>>> leftEye = Circle(Point(80, 50), 5)
>>> leftEye.setFill('yellow')
>>> leftEye.setOutline('red')
>>> rightEye = leftEye.clone() # rightEye is an
exact copy of the left
>>> rightEye.move(20, 0)
```

The String Data Type

Operator	Meaning
+	Concatenation
*	Repetition
<string>[]	Indexing
<string>[:]	Slicing
len(<string>)	Length
for <var> in <string>	Iteration through characters
<string>.split([var])	Get list of substrings seperated by char “var” (space by default)

Strings, Lists, and Sequences

- Strings are always sequences of characters, but *lists* can be sequences of arbitrary values
- Lists can have numbers, strings, or both

```
myList = [1, "Spam ", 4, "U"]
```

Input/Output as String Manipulation

```
print ("The converted date is:", end="")
print (monthStr, dayStr+",", yearStr)

>>> main()
Enter a date (mm/dd/yyyy): 01/23/2010
The converted date is: January 23, 2010
```

Files: Multi-line Strings

- A ***file*** is a sequence of data that is stored in secondary memory (disk drive)
- Files can contain any data type, but the easiest to work with are text
- Files usually contain multiple lines of text
- Python uses the standard newline character (**\n**) to mark line breaks

Multi-Line Strings

- **`Hello`**
  `World`

  `Goodbye 32`
- When stored in a file:
  `Hello\nWorld\n\nGoodbye 32\n`

Multi-Line Strings

- This is exactly the same thing as embedding **\n** in print statements
- Remember, these special characters only affect things when printed. They **don't do anything during evaluation**

File Processing

- The process of ***opening*** a file involves associating a file on disk with an object in memory
- We can manipulate the file by manipulating this object
  - **Read from the file**
  - **Write to the file**

File Processing

- When done with the file, it needs to be ***closed***. Closing the file causes any outstanding operations and other bookkeeping for the file to be completed
- In some cases, not properly closing a file could result in data loss

File Processing

Reading a file into a word processor

1. File opened
2. Contents read into RAM
3. File closed
4. Changes to the file are made to the copy stored in memory, not on the disk

File Processing

Saving a word processing file

1. The original file on the disk is reopened in a mode that will allow writing (this actually erases the old contents)
2. File writing operations copy the version of the document in memory to the disk
3. The file is closed

File Processing

- Working with text files in Python
  - Associate a disk file with a file object using the open function
    <filevar> = open(<name>, <mode>)
  - Name is a string with the actual file name on the disk. The mode is either 'r' or 'w' depending on whether we are reading or writing the file
  - `Infile = open("numbers.dat", "r")`

File Methods

- **<file>.read()** – returns the entire remaining contents of the file as a single (possibly large, multi-line) string
- **<file>.readline()** – returns the next line of the file. This is all text up to *and including* the next newline character
- **<file>.readlines()** – returns a list of the remaining lines in the file. *Each list item is a single* line including the newline characters

File Processing

```
# printfile.py
#     Prints a file to the screen.

def main():
    fname = input("Enter filename: ")
    infile = open(fname,'r')
    data = infile.read()
    print(data)

main()
```

- First, prompt the user for a file name
- Open the file for reading
- The file is read as one string and stored in the variable data

File Processing

- readline can be used to read the next line from a file, including the trailing newline character

```
infile = open(someFile, "r")
for i in range(5):
  line = infile.readline()
  print line[:-1]
```

- This reads the first 5 lines of a file
- Slicing is used to strip out the newline characters at the ends of the lines

File Processing

- Another way to loop through the contents of a file is to read it in with readlines and then loop through the resulting list.

```
infile = open(someFile, "r")

for line in infile.readlines():
   # Line processing here

infile.close()
```

File Processing

- Python treats the file itself as a sequence of lines!

```
Infile = open(someFile, "r")

for line in infile:
   # process the line here

infile.close()
```

File Processing

- Opening a file for writing prepares the file to receive data
- If you open an existing file for writing, you wipe out the file's contents. If the named file does not exist, a new one is created

```
Outfile = open("mydata.out", "w")

print(<expressions>, file=Outfile)
```

Example Program: Batch Usernames

- *Batch* mode processing is where program input and output are done through files (the program is not designed to be interactive)
- Example: create usernames for a computer system where the first and last names come from an input file

Example Program: Batch Usernames

```
# userfile.py
#    Program to create a file of usernames in batch mode.

def main():
    print ("This program creates a file of usernames from a")
    print ("file of names.")

    # get the file names
    infileName = input("What file are the names in? ")
    outfileName = input("What file should the usernames go in? ")

    # open the files
    infile = open(infileName, 'r')
    outfile = open(outfileName, 'w')
```

Example Program: Batch Usernames

```
# process each line of the input file
for line in infile:
    # get the first and last names from line
    first, last = line.split()
    # create a username
    uname = (first[0]+last[:7]).lower()
    # write it to the output file
    print(uname, file=outfile)

# close both files
infile.close()
outfile.close()

print("Usernames have been written to", outfileName)
```

Aside

- Non-text files are useful too

Intermission

http://cheezburger.com/42481409

```
import math

#4 - 4/3 + 4/5 - 4/7 + 4/9
def getApprox(terms):
    approx = 0

    for i in range(1, terms+1):
        approx = approx + 4/(2*i-1)*((-1)**(i-1))

    return approx

def getApprox2(terms):
    myList = [4, -4/3, 4/5, -4/7, 4/9]
    approx=0
    for i in range(0,terms):
        approx = approx + myList[i]
    return approx

def main():
    terms = eval(input("Num terms?"))
    approx = getApprox2(terms)
    print(approx, " ", math.pi - approx)
```