Programming in Python
Prog-what-ing?

A *program* is a *series of instructions* to a computer

```
00001f68 pushl $0x00
00001f6a movl %esp,%ebp
00001f6c andl $0xf0,%esp
00001f6f subl $0x10,%esp
00001f72 movl 0x04(%ebp),%ebx
00001f75 movl %ebx,0x00(%esp)
00001f79 leal 0x08(%ebp),%ecx
00001f7c movl %ecx,0x04(%esp)
00001f80 addl $0x01,%ebx
00001f83 shll $0x02,%ebx
00001f86 addl %ecx,%ebx
00001f88 movl %ebx,0x08(%esp)
00001f8c movl (%ebx),%eax
00001f8e addl $0x04,%ebx
00001f91 testl %eax,%eax
00001f93 jne 0x00001f8c
00001f95 movl %ebx,0x0c(%esp)
00001f99 calll 0x00001fca
00001f9e movl %eax,0x00(%esp)
00001fa2 calll 0x00003000
00001fa7 hlt
```
Prog-what-ing?

A **program** is a **series of instructions** to a computer

```
00001f68    pushl  $0x00
00001f6a    movl   %esp,%ebp
00001f6c    andl   $0xf0,%esp
00001f6f    subl   $0x10,%esp
00001f72    movl   0x04(%ebp),%ebx
00001f75    movl   %ebx,0x00(%esp)
00001f79    leal   0x08(%ebp),%ecx
00001f7c    movl   %ecx,0x04(%esp)
00001f80    addl   $0x01,%ebx
00001f83    shll   $0x02,%ebx
00001f86    addl   %ecx,%ebx
00001f88    movl   %ebx,0x08(%esp)
00001f8c    movl   (%ebx),%eax
00001f8e    addl   $0x04,%ebx
00001f91    testl  %eax,%eax
00001f93    jne   0x00001f8c
00001f95    movl   %ebx,0x0c(%esp)
00001f99    calll  0x00001fca
00001f9e    movl   %eax,0x00(%esp)
00001fa2    calll  0x00003000
00001fa7    hlt
```
Prog-what-ing?

A *program* is a *series of instructions* to a computer

```
00001f68 pushl  $0x00
00001f6a movl  %esp,%ebp
00001f6c andl  $0xf0,%esp
00001f6f subl  $0x10,%esp
00001f72 movl  0x04(%ebp),%ebx
00001f75 movl  %ebx,0x00(%esp)
00001f79 leal  0x08(%ebp),%ecx
00001f7c movl  %ecx,0x04(%esp)
00001f80 addl  $0x01,%ebx
00001f83 shll  $0x02,%ebx
00001f86 addl  %ecx,%ebx
00001f88 movl  %ebx,0x08(%esp)
00001f8c movl  (%ebx),%eax
00001f8e addl  $0x04,%ebx
00001f91 testl  %eax,%eax
00001f93 jne   0x00001f8c
00001f95 movl  %ebx,0x0c(%esp)
00001f99 calll  0x00001fca
00001f9e movl  %eax,0x00(%esp)
00001fa2 calll  0x00003000
00001fa7 hlt
```

A *programming language* makes this easier for humans
x = 10
y = 12.314
Less Headache - Python

```python
x = 10
y = 12.314
z = "Four score and seven..."

foo = x * y
print x
print y
print z
face = foo * y
print y, face
```
x = 10
y = 12.314
z = "Four score and seven..."
foo = x * y

print x
print y
print z

face = foo * y
print y, face
x = 10
y = 12.314
z = "Four score and seven..."
foo = x * y
print x
print y
print z
x = 10
y = 12.314
z = "Four score and seven..."
foo = x * y
print x
print y
print z
face = foo * y
print y, face
Demo Time!
= in gory detail

The variable on the left is assigned the value of the thing on the right

\[ x = 10 + 23 \]
= in gory detail

The variable on the left is assigned the value of the thing on the right

\[ x = 10 + 23 \]

First, calculate this
= in gory detail

The **variable** on the left is assigned the **value** of the **thing** on the right

\[ x = 10 + 23 \]

First, calculate this

and then give this that value

You can think of this as going from right to left.
Miniature “Quiz”

\[ x = 5 + 10 \]

\[ x = 3 \]
\[ y = x \times x \]

\[ 5 = 7 \times 3 \]

\[ abc = z - 11.2 \]
Miniature “Quiz”

\[ x = 5 + 10 \]  Yes!

\[ x = 3 \]
\[ y = x \times x \]  Yes!

\[ 5 = 7 \times 3 \]  No!

\[ abc = z - 11.2 \]  No! (subtle!)
If statements

variable1 = 10
variable2 = 12
variable3 = 0
if variable1 == variable2:
    variable3 = 11
else:
    variable3 = 12
print variable3

(== determines if two things are equal)
If-statements in gory detail

if something that might be true:
    x = 10
    y = "Krzyzewski"
    ...as many lines as you want...
elif something else that might be true:
    z = 12 * 13.0
else:
    print 12
    abc = 11.4
    z = 11
print z
If-statements in gory detail

```python
if something that might be true:
    x = 10
    y = "Krzyzewski"
    ...as many lines as you want...
elif something else that might be true:
    z = 12 * 13.0
else:
    print 12
    abc = 11.4
    z = 11
print z
```
If-statements in gory detail

```
if something that might be true:
    x = 10
    y = "Krzyzewski"
    ...as many lines as you want...
elif something else that might be true:
    z = 12 * 13.0
else:
    print 12
    abc = 11.4
    z = 11
    print z
```
If-statements in gory detail

```python
if something that might be true:
    x = 10
    y = "Krzyzewski"
    ...as many lines as you want...
elif something else that might be true:
    z = 12 * 13.0
else:
    print 12
    abc = 11.4
    z = 11
print z
```
If-statements in gory detail

```python
if something that might be true:
    x = 10
    y = "Krzyzewski"
    ...as many lines as you want...
elif something else that might be true:
    z = 12 * 13.0
else:
    print 12
    abc = 11.4
    z = 11
    print z
```
A trickier example: and / or

```python
face = 12
foo = "Four score"
temp = 3
if face > 6 or foo == 'seven years':
    temp = temp + 6
else:
    temp = temp + 4
if temp < 7 and foo == foo:
    face = face * temp
print face
```
A trickier example: and / or

```python
face = 12
foo = "Four score"
temp = 3  "is greater than"
if face > 6 or foo == 'seven years':
    temp = temp + 6
else:
    temp = temp + 4
if temp < 7 and foo == foo:
    face = face * temp
print face
```
single-quotes are ok too (but be consistent)
Project: Hello World

x = raw_input()
print "Alpha", "Zulu"

To Do:

Write a program that prompts you for your name and then says hello!
Loops

Print the numbers 1 through 100
Loops

Print the numbers 1 through 100

print 1
print 2
print 3
...
print 50
...
print 100
What we really want

x = 1
print x

# make x bigger by 1

x = x + 1   # make x bigger by 1
while x <= 100:
    print x
    x = x + 1
What we really want

```python
x = 1
print x
x = x + 1  # make x bigger by 1
```
What we really want

```
x = 1
print x
x = x + 1  # make x bigger by 1
```
What we really want

\[ x = 1 \]
\[ \text{print } x \]
\[ x = x + 1 \quad \# \text{ make } x \text{ bigger by } 1 \]

\[ x = 1 \]
\[ \text{while } x <= 100: \]
\[ \quad \text{print } x \]
\[ \quad x = x + 1 \]
Functions

\[ x = \text{abs}(-3.7) \]

- return value
- function name
- function argument

-3.7 \hspace{1cm} \text{abs} \hspace{1cm} \text{return value out}
Functions

\[ x = \text{abs}(-3.7) \]

- Function name: `abs`
- Return value: `3.7`
- Argument: `-3.7`
Functions

```
x = abs(-3.7)
```

- **Function Name**: abs
- **Function Argument**: -3.7
- **Return Value**: 3.7

The function `abs` takes an argument and returns its absolute value. (function does some work)
import math

```
cosine = math.cos(3.14)
```

```
help(math.cos)  # Provides help!
```

```
v = math.pow(2, 3)
```

*Multiple arguments!*
import math

cosine = math.cos(3.14)

help(math.cos)  # Provides help!

v = math.pow(2, 3)

x = 'Duke'
y = x.swapcase()
print y

Multiple arguments!

No arguments!
Things we have

- Variables and assignment
- *Conditionals* (if statements)
- Mathematical operations
- Repetition (loops)
One more thing

\[
x = 3 \\
y = 4 \\
\text{print } x / y
\]
One more thing

\[
\begin{align*}
x &= 3 \\
y &= 4 \\
\text{print } x / y
\end{align*}
\]

What happened?
One more thing

x = 3
y = 4
print x / y

What happened?

x = 3
y = 4
print float(x) / y

Better!
One more thing

```python
x = 3
y = 4
print x / y
```

What happened?

```python
x = 3
y = 4
print float(x) / y
```

Better!

*just another function!*
# Project: Magic π Trick

```python
# Put these three lines at the top of your file
import math
import random as R

# Useful math functions:
x = math.pow(2, 2)  # Power
y = math.sqrt(4)    # Square root

# Useful random-number functions:
z = R.random(0.0, 1.0)  # Get a number in the range [0, 1]
```

To Do:

- Calculate π using only what we’ve already seen and the functions above.