\[ \pi = \text{3.141592653589793...} \]

CIRCUMFERENCE OF A CIRCLE:

\[ 2\pi r^2 \]

\( r \) is the circle's radius

http://xkcd.com/10/
http://xkcd.com/1184/
Variables, types & flow control

Frédo Durand and Ana Bell
MIT EECS, 6.00
Bad news

• you cannot earn late days on pset 0
Grading

• Problem sets: 25%
• Quiz I: 15%  (March 6)
• Quiz II: 20% (April 17)
• Final: 35%
• Participation: 5%
Why Python?

easy   concise
-versatile, supports different styles
-widely used
Why Python?

• Easy syntax, concise
• Supports all styles of programming
• Widely used in science and engineering
  - lots of useful library
  - 6.815 is in Python!
• Easy to transfer knowledge to other languages

• Cons: can be slow
Recall PI
Imperative version of $\pi$

- Area of circle of radius 1
- Generate lots of points inside $[-1 \ 1]$ square
- Count point inside circle
  - they satisfy $x^2+y^2<1$
Computing $\pi$ in Python

```
N=1000.0
number_in_circle=0.0

x=-1.0
while x<1.0:
    y=-1.0
    while y<1.0:
        dist_square=x*x+y*y
        if dist_square<1:
            number_in_circle=number_in_circle+1
        y=y+1/N
    x=x+1/N

pi= number_in_circle/(N*N)
print pi
```
Computing $\pi$ in Python - comments

```
# We will sample points inside the square from -1, to 1 in both x and y
# We test which points inside the square are also inside the circle of radius 1.
# the ration should be $\pi/4$
# (the area of the square is 4, the area of the disk is $\pi$)
# note that we will space points 1/N apart, which means we will use 2Nx2N points

N=1000.0  #we will use 2Nx2N points
number_in_circle=0.0  #initialize counter of points inside the circle

x=-1.0  #we will start in the lower left corner, x=-1, y=-1
while x<1.0:  #loop over x, will stop when reaching the rightmost edge of the square
    y=-1.0
    while y<1.0:  #loop over y, will stop when reaching the top edge
        dist_square=x*x+y*y  #compute squared distance to the center
        if dist_square<1:  #if the (squared) distance is less than 1
            number_in_circle=number_in_circle+1  #then the point is in the circle, increment the counter
            y=y+1/N  #increment y to move to the next point
        x=x+1/N  #increment x to move to the next set of points

total_number_of_points=2*N*2*N  # because we spaced points 1/N apart and went from -1 to +1
pi= 4*number_in_circle/total_number_of_points  #the ratio of points inside the disk is $\pi/4$
print pi
```
Today

sequence of steps

variable

store & maintain information type

control flow
Today

• A program is a sequence of steps

• Variables
  - store and update values
  - have types

• Control flow with if and while
Python tutor

- [http://www.pythontutor.com/](http://www.pythontutor.com/)

- shows step by step execution
Variables
Math vs. CS

Math
\[ x, y, a \]

single letter ab \; axb

unknown, general numbers

CS
inside-circle i store information
1 piece of info at a time
Math vs. CS

• Math variable
  - alphabetic character representing a number which is either arbitrary or not fully specified or unknown
Math vs. CS

• Math variable
  - alphabetic character representing a number which is either arbitrary or not fully specified or unknown

• Computer science variable
  - adapted from http://en.wikipedia.org/wiki/Variable_(computer_science)
  - storage location and an associated symbolic name which contains some value
Variables for $\pi$ in Python

```python
N=1000.0
number_in_circle=0.0

x=-1.0
while x<1.0:
    y=-1.0
    while y<1.0:
        dist_square=x*x+y*y
        if dist_square<1:
            number_in_circle=number_in_circle+1
            y=y+1/N
            x=x+1/N
    x=x+1/N

pi= number_in_circle/(N*N)
print pi
```
Variables
Variables

• Store one value at a time

• Can be updated at any time
  - some variables don’t get updated much (N)
  - some change often
    (x, y, number_in_circle)
Names

• As meaningful as possible

• Convention: separate different words by _
  - for example: `number_in_circle`

• Learn auto-completion in IDLE
  - type beginning of name+option / on Mac

• Other possible convention (not in this class):
  – `numberInCircle`
Types
Type

integer  1  -1  2  3
float    1.1  -2.5  -2.0

x = 1.1
x = 2

string 'I am a string'
"I am a string"
"I'm a string"
$2 \geq 3$

$5 \geq -2.0$

Boolean:

True or False:

$x > 1$
Type

• Variables and expressions have types
• Numbers can be `int` (integers): 1, 2, -7, 1+1
• or `float` (approximation of reals): 2.2, -2.0, -1e6
• *Booleans* are `True` or `False` and represent the outcome of tests: 2>3 has value False
• *Strings* represent sets of characters, written between " or "": 'I am a string', "I am a string", 'a'

• More complex types later
Types in pi

N=1000.0
number_in_circle=0.0
x=-1.0
while x<1.0:
    y=-1.0
    while y<1.0:
        dist_square=x*x+y*y
        if dist_square<1:
            number_in_circle=number_in_circle+1
        y=y+1/N
    x=x+1/N
pi= number_in_circle/(N*N)
print 'pi is equal to ' 
print pi
\pi \text{ with boolean variable}

\begin{align*}
N &= 1000.0 \\
\text{number\_in\_circle} &= 0.0
\end{align*}

\begin{align*}
x &= -1.0 \\
\text{while } x < 1.0: \\
& \quad y = -1.0 \\
& \quad \text{while } y < 1.0: \\
& \quad \quad \text{dist\_square} = x \times x + y \times y \\
& \quad \quad \text{is\_in\_disk} = x \times x + y \times y < 1 \\
& \quad \quad \text{if is\_in\_disk:} \\
& \quad \quad \quad \text{number\_in\_circle} = \text{number\_in\_circle} + 1 \\
& \quad \quad y = y + 1/N \\
& \quad x = x + 1/N
\end{align*}

\begin{align*}
\pi &= \text{number\_in\_circle} / (N \times N) \\
\text{print } \pi
\end{align*}

\textit{I found init to False} \\
\textit{modify if you find stuff}
Type?

17  int
15.7  float
14.0  float
'hello world'  str
'15.0'  str
1+1  str
1/2  int
1/2.0  float
1>2  bool
Casting (type translation)

\[ 1 + 2.0 \rightarrow \text{float} \]

\[
\text{float} (2) \quad 1 / \text{float} (N)
\]

\[
\text{int} (2.4) \quad \text{int} ('a')
\]

\[
\text{int} ('2') \quad \text{int} ('1+1') \rightarrow \text{error}
\]
Casting (type translation)

- int get promoted (cast) to float in mixed expressions:
  - 1+2.0 returns a float, 3.0

- Casting can also be explicit:
  - float(3)
  - str(2)
  - int('2')
  - int('blah') → error
int vs. float - problem

```
N=1000  BAD
number_in_circle=0

x=-1.0
while x<1.0:
    y=-1.0
    while y<1.0:
        dist_square=x*x+y*y
        if dist_square<1:
            number_in_circle=number_in_circle+1
        y=y+1/N
    x=x+1/N

pi= number_in_circle/(N*N)
print pi
```
N=1000.0

number_in_circle=0.0

x=-1.0
while x<1.0:
    y=-1.0
    while y<1.0:
        dist_square=x*x+y*y
        if dist_square<1:
            number_in_circle=number_in_circle+1
        y=y+1/N
    x=x+1/N

pi= number_in_circle/(N*N)
print pi
Careful

\[ N = 2 \]

\[
\text{average} = \frac{(3 + 4)}{N} \quad \text{BAD}
\]

\[ N = 2.0 \]

or

\[
\left( \frac{3 + 4}{\text{float}(N)} \right)
\]

\[ 2C = 2C + 1/N \]
average = \frac{3 + 4}{N} \quad (BAD)

N = 2.0

or

\frac{3+4}{\text{float}(N)}

x = x + 1/N

\text{first compute this}

\text{second operation}
Careful

- It is easy to do write
  \[ N=2 \]
  \[ \text{average} = (3+2)/N \]
For the more curious

• You can ask the type of an expression (or variable) with type(expression)
  - for example: type(3)
  - test: type(N) == type(x)
Binding: the = and == challenge
Binding

\[ x = 1 + 2 \]

- Take RHS
- Compute value
- Bind it (store it)
- In LHS variable

\[ \_ = \_ \]
Binding

• = does not have the same meaning as in math
• x=2+3
  - compute the right-hand-side value
  and bind it to the left-hand side variable
  - binding means storing, assigning, replacing whatever was there

• 2+3=x does not mean anything in Python
  (=>error)
Math vs. CS:  \( x = x^x \)

**Math**

\[ x = x^2 \quad 0 \text{ or } 1 \]

**Python**

\[ x = 2 \]

\[ x = x \times x \quad \Rightarrow \quad x \text{ has value } 4 \]
Math vs. CS: $x=x^x$

- Math: $x$ is either 0 or 1
- CS: take whatever value $x$ used to have, multiply it by itself, store it in $x$.
  
  - for example:
    
    $x=3$
    
    $x=x^x$
    
    $x$ is now 9
Binding
Binding

• for example:

\[ N=3 \]
\[ N=N+1 \]
Binding

- for example:

\[
N = 3 \\
N = N + 1
\]

- The last line, \(N = N + 1\) involves the following
  - compute right-hand-side
    - get current value of \(N\): 3
    - add 1 to it, gives 4
  - bind \(N\) to 4
\begin{itemize}
  \item $y = 2$
  \item $x = 3$
  \item $x = y$

\item vs.

  \item $y = 2$
  \item $x = 3$
  \item $y = x$
\end{itemize}
=  

- x=1
  y=x
  x=2
  print x, y
swapping variables

\[
\begin{align*}
  x &= 2 \\
  y &= 3 \\
  x &= y \\
  y &= 2x \\
  \text{tmp} &= x \\
  x &= y \\
  y &= \text{tmp}
\end{align*}
\]
swapping variables

• start with $x=3; y=2$

• DO NOT WRITE:
  
  ```
  x=y
  y=x
  ```

• Instead
  
  ```
  tmp=x
  x=y
  y=tmp
  ```

- (Python also has a simpler way. More later.)
Test: $\equiv$

\[
1 \equiv 3 - 2
\]

operation returns Boolean
Test: ==

• if x==2: print ‘yes’
  - does the printing only if x has value 2

• == computes the value on both sides, then compares them.
  Returns true if they are the same, false otherwise

• This one is symmetric:

• if 2==x: print ‘yes’
  - does the same
Recap: The `=` vs. `==` challenge

• `=` in computer science is very different from `=` in math
• the left hand side of `=` is always a variable name
• `=` means that the result of the right expression gets stored in the variable
• some languages use `<-` or `:=` instead of `=` to be clearer
• to make matters worse, there is a test to see if two expressions are equal, and in Python it’s `==`
Input & output
Output: print
Output: print

- \( x=3 \)
  
  ```python
  print x
  show 3 on the console
  ```

- print followed by arguments (any type) separated by comas:
  
  - \( x=3 \)
    
    ```python
    print 'x=', x
    ```

  - note the string between ” (could also be ”x=”)

- Careful: don’t use Python 3, print doesn’t work the same
Input: raw_input

- \( x = \text{int(raw_input('give me a number '))} \)
Input: raw_input

- \( x = \text{int}(\text{raw\_input}(\text{'give me a number '})) \)
  - Shows \textit{give me a number}
  - waits until the user types a number and return
  - binds that number to the variable \( x \)

- raw\_input returns a string
  - this is why we need to cast to \textit{int},
  - returns error if the string is not a number
Operations
Operations

• numbers => number
  +, -, *, /, ** (exponent), % (remainder of integer division)

• numbers=>boolean
  ==, != (different), >, <, >=, <=

• boolean=>boolean
  and, not, or

• strings
  +, *,...
Operator precedence

• What you’d expect
• use parentheses when unsure
Control
Elements of a program

- Sequence of simple steps
- Store and update information in variables
- Flow of control (while, if)
While
While

while condition:

step1

step2

stepN

• where condition returns a boolean

• keeps going through the steps as long as condition is True
While - example
While - example

N=10

while N>0:
    N=N-1

print N outside the while
Spaces and indentation
Spaces and indentation

- Indentation matters a lot in Python
  - “group” things (called blocks)

- while condition:
  step1
  step2
  step3

- step 1 and 2 are in the loop, not step 3
- also note the :

- Extra carriage returns are optional but make the code easier to read
N=1000
number_in_circle=0.0

x=-1.0
while x<1.0:
    y=-1.0
    while y<1.0:
        dist_square=x*x+y*y
        if dist_square<1:
            number_in_circle=number_in_circle+1
        y=y+1/N
    x=x+1/N

pi= number_in_circle/(N*N)
print pi
If

if condition:
    step1
    stepN
If, else

if condition:
  step1
  stepN

if condition
  step1
  stepN

else:
  stepA
  stepM
If, else

If x%2==0:
    print ‘even’
else:
    print ‘odd’
x = int(raw_input('Enter an integer: '))
if x%2 == 0:
    print 'Even'
else:
    print 'Odd'
if x%3 != 0:
    print 'And not divisible by 3'
If, else

If x%2==0:
    print 'even'
else:
    print 'odd'
If, else

```python
if condition:
    step1
    stepN

if condition
    step1
    stepN
else:
    stepA
    stepM
```
If, else

If x%2==0:
    print 'even'
else:
    print 'odd'
If within if

```python
x = int(raw_input('Enter an integer: '))
if x%2 == 0:
    print 'Even'
else:
    print 'Odd'
    if x%3 != 0:
        print 'And not divisible by 3'
```
N=1000
number_in_circle=0.0
number_out_circle=0.0

x=-1.0
while x<1.0:
  y=-1.0
  while y<1.0:
    dist_square=x*x+y*y
    if dist_square<1:
      number_in_circle=number_in_circle+1
    else:
      number_out_circle=number_out_circle+1
    y=y+1/N
  x=x+1/N

pi= number_in_circle/(number_in_circle+number_out_circle)
print pi
Recap

• A program is a sequence of steps

• Variables
  - store and update values
  - have types

• Control flow with if and while