Introduction to Programming II
W4260
Lecture 2
Overview

• Storing Data
  – Basic types
  – Arrays

• Controlling the flow of execution
  – Loops (for, while)
  – If…then…else

• Operators
  – Arithmetic, relational, logical

• Functions (sub-program)
  – Built-in functions
  – Defining functions
Variables

• Name
  – e.g. x, y, index, NSync, Nigela6

• Types
  – Integer (-2147483648 to +2147483647)
  – Float (6 digits, $10^{-38}$ to $10^{+38}$)
  – Double (15 digits, $10^{-308}$ to $10^{+308}$)
  – Boolean (true or false)
  – Character or string
More on data storage: Arrays

- Arrays are collections of numbers
  - all numbers within an array have the same type (e.g. integer)

C program

```c
main()
{
    int primes[4];
    primes[0] = 1;
    primes[1] = 2;
    primes[2] = 3;
    primes[3] = 5;
}
```

PYTHON program

```python
from numpy import *
primes = zeros(4, int)
primes[0] = 1
primes[1] = 2
primes[2] = 3
primes[3] = 5
```
More on data storage: Arrays

• Array contents can be modified

C program

```c
main()
{
    int primes[4];
    primes[0] = 1;
    primes[1] = 2;
    primes[2] = 3;
    primes[3] = 5;
}
```

PYTHON program

```python
from numpy import *
primes = zeros(4, int)
primes[0] = 1
primes[1] = 2
primes[2] = 3
primes[3] = 5
```
Why use arrays?

- Store a list of measurements
  - Can then compute mean, etc
- Vectors and matrices

\[
\begin{pmatrix}
1 & 5 & 2 \\
2 & 1 & 5 \\
-3 & 11 & 9
\end{pmatrix}
\begin{pmatrix}
2 \\
1 \\
3
\end{pmatrix}
= ?
\]

C program
```c
main()
{
    double vector[3];
    double matrix[3][3];

    matrix[0][0] = 1.0;
    matrix[0][1] = 5.0;
    ...
}
```

PYTHON program
```python
from numpy import *

vector = zeros(3)
matrix = zeros([3,3])

matrix[0,0] = 1.0
matrix[0,1] = 5.0
...
```
Part B: Flow of Control

1. Loops
2. Branches (if...then...else)
Flow of control

• Top to bottom (within a program) unless:

1. Loops
   - for loops
   - while loops

2. Branches
   - If...then...else

3. Functions
1. Loops

For more strength and shine use the rest of the Classic Care Collection - Shampoo, Conditioner and Moisturising Masque.

**DIRECTIONS:** Lather, rinse, repeat if required.
1(a). While loops

• allows for repeated actions

while (condition is true)
    {do this}

while condition is true:
    do this

C program

```c
main()
{
    float x, y;
    x=0.1;
    while (x < 2.0) {
        y = x*x;
        printf("%f %f\n", x, y);
        x = x + 0.1;
    }
}
```

PYTHON program

```python
x=0.1
while x < 2.0:
    y = x*x
    print x, y
    x = x + 0.1
```
1(a). While loops

- allows for repeated actions

Repeated statements
surrounded by braces {}

C program

```c
main()
{
    float x, y;

    x=0.1;
    while (x < 2.0) {
        y = x*x;
        printf("%f %f\n", x, y);
        x = x + 0.1;
    }
}
```

Repeated statements
indented

PYTHON program

```python
x=0.1
while x < 2.0:
    y = x*x
    print x, y
    x = x + 0.1
```
1(b). for loops

- short-hand way to loop for a certain number of times

C program

```c
main()
{
    int i;

    for (i=0; i < 5; i++) {
        printf("%d bottles of beer\n",i);
    }
}
```

PYTHON program

```python
for i in range(0,5):
    print "%d bottles of beer" % i
```
1(b). for loops

• short-hand way to loop for a certain number of times

C program

```c
main()
{
  int i;

  for (i=0; i < 5; i++) {
    printf("%d bottles of beer\n",i);
  }
}
```

PYTHON program

```python
for i in range(0,5):
    print "%d bottles of beer" % i
```

Equivalent to [0,1,2,3,4]
Looping in $\pi$ program

$$\pi = 2\sqrt{3}\left(1 - \frac{1}{3 \times 3} + \frac{1}{5 \times 3^2} - \frac{1}{7 \times 3^3} + \ldots\right)$$

C program

```c
#include <math.h>

main()
{
    double sum, pi;
    int i;
    sum = 1.0;
    for (i = 1; i < 30; i = i + 1)
        sum = sum+1.0/((i*2+1)*pow(-3.0,i));

    pi = 2.0*sqrt(3.0)*sum;

    printf("%.20f\n", 2.0*sqrt(3.0)*sum);
}
```

Python program

```python
from math import *

sum = 1.0

for i in range(1,30):
    sum = sum+1.0/((i*2+1)*(-3.0)**i)

pi = 2.0*sqrt(3.0)*sum

print "% .20f\n" % pi
```
2. Branching (if statement)

- allows for different actions depending on input

```c
main()
{
    ...
    if (x < 2.0) {
        printf("x is less than 2!\n");
    }
}
```

```python
... if x < 2.0:
    print "x is less than 2!"
```
2. Branching (if...else statement)

- allows for different actions depending on input

if (condition is true)
    {do this}
else
    {do that}

C program

```c
main()
{
    ...
    if (x < 2.0) {
        printf("x is less than 2!\n");
    } else {
        printf("x is >= 2!\n");
    }
}
```

PYTHON program

```python
...
if x < 2.0:
    print "x is less than 2!"
else:
    print "x is >= 2!"
```
Can combine ("nest") loops

C program

```c
main()
{
    int i;

    for (i=0; i < 5; i++) {
        if (i == 1)
            printf("1 bottle of beer\n");
        else
            printf("%d bottles of beer\n", i);
    }
}
```

PYTHON program

```python
for i in range(0,5):
    if (i == 1):
        print "1 bottle of beer"
    else:
        print "%d bottles of beer" % i
```
Can combine ("nest") loops

C program

```c
main()
{
    int i;

    for (i=0; i < 5; i++) {
        if (i == 1)
            printf("1 bottle of beer\n");
        else
            printf("%d bottles of beer\n", i);
    }
}
```

PYTHON program

```python
for i in range(0,5):
    if (i == 1):
        print "1 bottle of beer"
    else:
        print "%d bottles of beer" % i
```

Dropped braces because only 1 statement

Note: relational operator for equals is `==`, not `=`
Summary

• Various kinds of variables (int, float, …)
• Float/double have limited precision
• Arrays are useful ways to group numbers
• Loops
  – While, for
• If…then…else
What is an operator?
What is an operator?
What is an operator?

• Mathematical: \( L[f(x)] = g(x) \)
  – \( L \) might be \( \frac{d}{dx} \)
  – \( L[\sin(x)] = \cos(x) \)

• Computational:

```
  (unary) operator
  \[\text{datum} \rightarrow \text{datum}\]
  or
  (binary) operator
  \[\text{datum} \rightarrow \text{datum} \rightarrow \text{datum}\]
```
The 3 Types of Operators

(1) arithmetic operator
    (binary/unary)
    number
    number
    e.g. 1+3 → 4

(2) relational operator
    (binary only)
    number
    number
    e.g. 1<3 → true

(3) logical operator
    (binary/unary)
    boolean
    boolean
    e.g. true OR false
    ↓ true
1. Arithmetic operators
1. Arithmetic operators

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Exponential: **
- Modulo: %
- Grouping: ()

Examples:
- 3 + x
- y - 3 * x
- (2.3 / y + 5) * x
- 2.0 ** x
1. Arithmetic operators

Two arguments of the same type produce a result with that type

float

float

arithmetic operator

float

2.0/4.0 → 0.5
1. Arithmetic operators

Two arguments of different types make the result the higher precision type.

2.0/4.0 → 0.5

2.0/4 → 0.5
1. Arithmetic operators

Warning: integer arithmetic can cause unexpected results

integer operator integer

2/4 → 0
1. Arithmetic operators: Example

```
x=0.1
While x < 2.0:
    y = x*x
    print x, y
    x = x + 0.1
```

**Python program**

**Binary operation: multiply**

- **In:** x, x
- **Out:** x*x

**Binary operation: add**

- **In:** x, 0.1
- **Out:** x+0.1
2. Relational operators
2. Relational operators
2. Relational operators

Relational operators compare two values and return a boolean result.

- `<` less than
- `<=` less than or equal
- `>` greater than
- `>=` greater than or equal to
- `==` equal to
- `!=` not equal to

Example:
- `3 < 4`
- `x != (4+x*y)`
- `index == i*2`

Diagram:

- `number` ➔ `relational operator` ➔ `boolean` (true or false)
2. Relational operators: Example

... if \( n == 3 \):
    \[ x = \frac{4}{3}\pi r^3 \]

if \( n == 2 \):
    \[ x = \pi r^2 \]

if \( n == 1 \):
    \[ x = 2\pi r \]

if \( n > 3 \):
    print "n is too large"

...
2. Relational operators: warning

Note: difference between `==` and `=`

Relational operator: equal-to

```
if n == 3:
```

Variable assignment

```
n = 3
```

... 
```
n = 3
... 
```

```
if n == 3:
    x = 4/3*pi*r*r*r
```

This changes the value of `n`

This doesn’t

Warning: confusing the two can cause difficult to debug errors.
3. Logical operators
3. Logical operators
3. Logical operators

C

<table>
<thead>
<tr>
<th>&amp;&amp;</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

e.g. \( x < 3 \) && \( y > 4 \)
\( (i-4) == 1 \) || \( x != 2 \)

Python

<table>
<thead>
<tr>
<th>and</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td>OR</td>
</tr>
</tbody>
</table>

e.g. \( x < 3 \) and \( y > 4 \)
\( (i-4) == 1 \) or \( x != 2 \)
3. Relational operators: Example

```python
... 
if n == 3:
    x = 4/3*pi*r*r*r
if n == 2:
    x = pi*r*r
if n == 1:
    x = 2*pi*r
if n > 3:
    print "n is too large"
... 
```

What if n = 0?
3. Relational operators: Example

```
... if n == 3:
    x = 4/3*pi*r*r*r
if n == 2:
    x = pi*r*r
if n == 1:
    x = 2*pi*r
if n < 1 or n > 3:
    print "n is out of range"
...```

B. Functions

• Math:
  – define $f(x) = x^2$
  – $f(3) = 9$
  – define $g(x,y) = (x^2 + y^2)^{1/2}$
  – $g(3,4) = 5$
Functions are sub-programs

• Take any number of arguments
  – Any types

• Returns one value
  – Any type

result = func(arg1, arg2, ...)

1. Built-in functions

- “built-in” to language (library function)
- many math functions, string manipulations

C program
```
#include <math.h>

main()
{
    double x, y;
    x = 3.0;
    y = sqrt(x);
}
```

PYTHON program
```
from math import *

x = 3.0
y = sqrt(x)
```

“Imports” the standard library of math functions
2. Writing your own functions

1. Define function
2. Invoke it from main program

```c
#include <math.h>

float radius(double a, double b) {
    return sqrt(a*a + b*b);
}

main() {
    double x, y;
    x = 3.0;
    y = radius(x, 4.5);
}
```

```python
from math import *

def radius(a, b):
    return sqrt(a*a + b*b)

x = 3.0
y = radius(x, 4.5)
```
2. Writing your own functions

1. Define function
2. Invoke it from main program

C program

```c
#include <math.h>

float radius(double a, double b) {
    return sqrt(a*a + b*b);
}

main() {
    double x, y;
    x = 3;
    y = radius(x, 4.5);
}
```

PYTHON program

```python
from math import *

def radius(a, b):
    return sqrt(a*a + b*b)

x = 3.0
y = radius(x, 4.5)
```
A prime number algorithm

Step 1: List out the numbers from 1 to n
Step 2: Strike out all numbers divisible by 2 (except 2)
        Strike out all numbers divisible by 3 (except 3)
        and so on for 7, 11, 13...
Step 3: List out the remaining numbers, they are prime.

1  2  3  4  5  6  7  8  9  10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
...


primes: The program

### C program

```c
main()
{
    int numbers[1000], i, j, m, n;
    n = 1000;
    m = sqrt(n)+1;

    for (i=1; i < n; i++)
        numbers[i] = i;

    for (j=2; j < m; j++)
        if (numbers[j] != 0)
            for (i=2*j; i < n; i=i+j)
                numbers[i] = 0;

    for (i=1; i < n; i++)
        if (numbers[i] != 0)
            printf("%d
", numbers[i]);
}
```

### PYTHON program

```python
def main():
    numbers = range(0,1000)
    n = 1000
    m = sqrt(n)+1

    for i in range(1,n):
        numbers[i] = i

    for j in range(2, m):
        if numbers[j] != 0:
            for i in range(2*j,n,j):
                numbers[i] = 0

    for i in range(1,n):
        if numbers[i] != 0:
            print(numbers[i])
```
1. Storage of data

- variables store data
- different types (integer, real, text, …)

```
C program

main()
{
    int numbers[1000], i, j, m, n;
    n = 1000;
    m = sqrt(n)+1;

    for (i=1; i < n; i++)
        numbers[i] = i;
...

PYTHON program

numbers = range(0,1000)
n = 1000
m = sqrt(n)+1

for i in range(1,n):
    numbers[i] = i;
...
```
1. Storage of data

- arrays

C program

```c
main()
{
    int numbers[1000], i, j, m, n;
    n = 1000;
    m = sqrt(n)+1;

    for (i=1; i < n; i++)
        numbers[i] = i;

    ... 
}
```

PYTHON program

```python
numbers = range(0, 1000)
n = 1000
m = sqrt(n)+1

for i in range(1,n):
    numbers[i] = i;

...
```
2. (Arithmetic) Operations

- elementary maths (+, -, multiply, divide)
- other operations (library functions)

\[
\begin{align*}
n &= 1000 \\
m &= \sqrt{n} + 1 \\&= 32
\end{align*}
\]

C program
```c
main()
{
    int numbers[1000], i, j, m, n;
    n = 1000;
    m = sqrt(n) + 1;

    for (i=1; i < n; i++)
        numbers[i] = i;

    ...
}
```

PYTHON program
```python
numbers = range(0,1000)
n = 1000
m = sqrt(n) + 1

for i in range(1,n):
    numbers[i] = i;

...
```
3. Flow of Control: Loops

- perform (nearly) the same operation many times

```c
main()
{
    int numbers[1000], i, j, m, n;
    n = 1000;
    m = sqrt(n)+1;
    for (i=1; i < n; i++)
        numbers[i] = i;
...
```

```python
numbers = range(0,1000)
n = 1000
m = sqrt(n)+1
for i in range(1,n):
    numbers[i] = i;
...
```
More loops…

Step 2: strike out all numbers divisible by 2  (j=2)
strike out all numbers divisible by 3  (j=3)
strike out all numbers divisible by 4  (j=4)
strike out all numbers divisible by 5  (j=5)
...

<table>
<thead>
<tr>
<th>C program</th>
<th>PYTHON program</th>
</tr>
</thead>
</table>
| for (j=2; j < m; j++)
  if (numbers[j] != 0)
    for (i=2*j; i < n; i=i+j)
      numbers[i] = 0;
| for j in range(2, m):
  if numbers[j] != 0:
    for i in range(2*j,n,j):
      numbers[i] = 0
  ... |
More loops...

Step 2: strike out all numbers divisible by 2 (excluding 2 itself)
4, 6, 8, 10...

C program

```c
... for (j=2; j < m; j++)
    if (numbers[j] != 0)
        for (i=2*j; i < n; i=i+j)
            numbers[i] = 0;
...```

PYTHON program

```python
... for j in range(2, m):
    if numbers[j] != 0:
        for i in range(2*j, n, j):
            numbers[i] = 0
...```
4. Flow of Control: If statement

Step 2: strike out all numbers divisible by 2  (j=2)
strike out all numbers divisible by 3  (j=3)
strike out all numbers divisible by 4  (j=4) unnecessary
strike out all numbers divisible by 5  (j=5)
...

numbers: 1 2 3 0 5 0 7 ...

C program

...  
for (j=2; j < m; j++)
   if (numbers[j] != 0)
      for (i=2*j; i < n; i=i+j)
         numbers[i] = 0;
...  

PYTHON program

...  
for j in range(2, m):
   if numbers[j] != 0:
      for i in range(2*j, n, j):
         numbers[i] = 0
...  

4b. An alternative: functions

Can define a function (strike_out) which actually does the striking out of one multiple (e.g. all the 2’s)

C program

```c
strike_out(int numbers[], int n, int j) {
    for (i=2*j; i < n; i = i + j)
        numbers[i] = 0;
}
...
for (j=2; j < m; j++)
    if (numbers[j] != 0)
        strike_out(numbers, n, j);
...```

PYTHON program

```python
def strike_out(numbers, n, j):
    for i in range(2*j, n, j):
        numbers[i] = 0;
...
for j in range(2, m):
    if numbers[j] != 0:
        strike_out(numbers, n, j)
...```
5. Output (Step 3)

C program

```c
main()
{
    int numbers[1000], i, j, m, n;
    n = 1000;
    m = sqrt(n)+1;

    for (i=1; i < n; i++)
        numbers[i] = i;

    for (j=2; j < m; j++)
        if (numbers[j] != 0)
            for (i=2*j; i < n; i=i+j)
                numbers[i] = 0;

    for (i=1; i < n; i++)
        if (numbers[i] != 0)
            printf("%d\n", numbers[i]);
}
```

PYTHON program

```python
numbers = range(0,1000)
n = 1000
m = sqrt(n)+1

for i in range(1,n):
    numbers[i] = i

for j in range(2, m):
    if numbers[j] != 0:
        for i in range(2*j,n,j):
            numbers[i] = 0

for i in range(1,n):
    if numbers[i] != 0:
        print numbers[i]
```
Summary

• Operators
  – arithmetic, relational, logical

• Functions
  – Built-in
  – Define yourself
    • Do not even have to return a value