Python in COMP[29]041

We will introduce **briefly** Python in lectures after Perl and compare it to Perl.
May be a written exam question on Python.
Challenge part of lab exercises involves Python.
Opportunity for student coping **well** with Perl to teach themselves Python.
Assignments involve Python.

Python

Developed in the 1990s by Guido van Rossum at CWI (Netherlands) - was at Google - now at Dropbox.
Similar niche to Perl but very different design.
Much simpler syntax than Perl.
No variable interpolation in strings, use % operator instead
Regular expressions used via functions, not embedded in syntax.
Indenting used to group statements - unlike Perl/C/Java/...
Python has a more elaborate type system but more straightforward semantics.
Python (like C) does less implicit conversions than Perl, e.g. you have to convert strings explictly to numbers.
Core python does not have arrays (numpy has arrays).
Python lists can be indexed but do not automatically grow.

Python 2 or Python 3

Python introduced in 2008 but many users stayed with Python 2 until recently.
Some users still use Python 2 - some useful packages not yet available for Python 3.
Mostly possible to write programs which work in both Python 2 and 3.
Most COMP[29]041 example code will work in both Python 2 and 3.
You may use either Python 2 or Python 3 in COMP[29]041.

Syntax Conventions

Perl uses non-alphabetic characters such as $, @, % and & to associate types with names.
Python associates types with values.

Perl	**Python**
`$s = "string"`	`s = "string"`
`@a = (1,2,3)`	`a = [1,2,3]`
`%a = (1=>’a’,2=>’b’,3=>’c’)`	`a = {1:’a’,2:’b’,3:’c’}`
`$a[42] = 'answer'`	`a[42] = 'answer'`
`$h{'answer'} = 42`	`h['answer'] = 42`

Names and Types

A Python name can be associated with any type.
The **type** function allows introspection.

```
>>> a = 42
>>> type(a)
<type 'int'>
>>> a = "String"
>>> type(a)
<type 'str'>
>>> a = [1,2,3]
>>> type(a)
<type 'list'>
>>> a = {'ps':50,'cr':65,'dn':75}
>>> type(a)
<type 'dict'>
```

Python Indexing

Indexing is generalized in Python.
Python uses [] to access elements of both lists and dicts (hashes).
[] also used for other types (e.g. strings).
[] can be used for user-defined class if it has __**getitem**__ method.

Initializing dicts

In Perl you can assign a value to hash element without having used the hash previously:

```
$h{'answer'} = 42
```

In Python you must first initialize the dict: like this:

```
h = {}
h['answer'] = 42
```

or you could do this:

```
h = {'answer':42}
```

Python lists

Perl arrays grow automatically - Python lists do not. This fails in Python:

```
h = []
h[0] = 'zero'
```

This works:

```
h = []
h.append('zero')
```

or this works:

```
h = ['zero']
```

A useful idiom to create a large list of constants is:

```
h = [0] * 100
```

Variable Interpolation

Python does not interpolate variables into strings.
There is no difference between single and double quotes.
Python has the % operator - equivalent to sprintf.
So you can write:

```
    print('%8d %s' % (count[word], word))
```

See also string.Template

Regular expressions

Regular expressions not built-in to Python syntax
Less concise than Perl but similar functionality.

```
$x =~ s/a/b/;
```

```
x = re.sub('a', 'b', x)
```

Python r-quotes useful for regex - if a string is prefixed with an r (for raw) then backslashes have no special meaning (except before quotes).
In Python **r'\n'** is a 2 character string as is **'\n'** in Perl.

Defining Python Functions

Python functions have named untyped parameters:

```
def square(x):
    return x*x
```

Function calls are C-style, e.g:

```
print(square(42))
```

Default argument values can be indicated:

```
def square(x=42):
        return x*x
```

Allowing the function to be optionally called without specifying the argument.

```
print(square())
```

Keyword Functions Arguments

Function arguments can be specified by keyword.
This funcion:

```
def evaluate_poly(a=1, b=2, c=0, x=42):
    return a*x*x+b*x+c
```

Can be called like this:

```
print(evaluate_poly(x=7, c=4))
```

Accessing Function Arguments as List or Dict

Can also have variable number of non-keywords arguments accesssed by a list.
And a variable number of non-keywords arguments accesssed by a dict.

```
def f(*arguments, **keywords):
    for arg in arguments:
        print(arg)
    for kw in keywords.keys():
        print(kw + " : " + keywords[kw])
```

Powerful but you don't need for COMP2041.

Python imports

Python module - a file containing function definitions and other Python.
Access modules using import statements,e.g.
import random can access names from random as random.name, e.g **random.choice**
from random import shuffle can access **random.shuffle** as shuffle
from random import * can access all names from random without prefix
import random as r access names from random as r.name, e.g **r.choice**

Perl-Python Comparison: Counting First names

```
while ($line = <>) {
    @fields = split /\|/, $line;
    $student_number = $fields[1];
    next if $already_counted{$student_number};
    $already_counted{$student_number} = 1;
    $full_name = $fields[2];
    $full_name =~ /.*,\s+(\S+)/ or next;
    $fname = $1;
    $fn{$fname}++;
}

foreach $fname (sort keys %fn) {
    printf "%2d people with the first name $fname\n", $fn{$fname
}
```

Perl-Python Comparison: Counting First names

```python
import fileinput, re
already_counted = {}; fn = {}
for line in fileinput.input():
    fields = line.split('|')
    student_number = fields[1]
    if student_number in already_counted: continue
    already_counted[student_number] = 1
    full_name = fields[2]
    m = re.match(r'.*,\s+(\S+)', full_name)
    if m:
        fname = m.group(1)
        if fname in fn:
            fn[fname] += 1
        else:
            fn[fname] = 1
for fname in sorted(fn.keys()):
    print("%2d people with first name %s"%(fn[fname], fname))
```