Name: _____

**Directions: Work only on this sheet (on both sides, if needed); do not turn in any supplementary sheets of paper. There is actually plenty of room for your answers, as long as you organize yourself BEFORE starting writing.**

**Do not use any Python or Perl constructs which were not introduced either in lecture, discussion section or our written materials.**

**1.** (10) Give Python analogs of the following Perl items: **shift @z**, **unshift @z,$x** (prepends), **$ARGV[3]**, **@z[9:12]**.

**2.** (15) Add code to **kill()** in our PLN program **psax.py**, which will display a confirmation query message, "Kill this process (y,n)?", on the line which is highlighted at the time the user hits the k key. The message will overwrite the first part of that line. If the next key hit is not y, the function will simply return (the user can then hit u etc. as a means of restoring the line); otherwise the original line will be restored and the process will be killed. In the latter case, change the State column ($16^{th}$ character position in the line) to 'K'. You are allowed to add a total of nine lines maximum.

**3.** Consider a package **RadixNum.pm** whose objects will store nonnegative numbers in base-r form internally but will present a scalar version of the numbers to the "outside world." For example, consider this code (ignore the three blank lines at the end for now):

```
use RadixNum;

tie $x,'RadixNum',3,0;  # base 3
tie $y,'RadixNum',3,0;  # base 3
$x = 5;
$y = $x + 20;
print $x,$y, "\n";
------------------------
------------------------
------------------------
```

This will print out 5 and 25, but internally **$y**, for instance, will also be associated with a reference to an anonymous array (2,2,1), since the base-3 representation of 25 is $\mathbf{2} \cdot 3^2 + \mathbf{2} \cdot 3^1 + \mathbf{1} \cdot 3^0$. Note that the fancy name used for base is *radix*, so in this example the radix is 3.

The object created by **TIESCALAR()** (not shown) is an anonymous hash which consists of an integer **Radix**, a reference **Digits** to the anonymous array, and an integer **Value** containing the numeric value.

Part of the code in **RadixNum.pm** will be a subroutine **torad()**. It has as arguments the radix and value, and returns a reference to the corresponding radix form of the value. For instance, the call **torad(3,25)** will return \(2,2,1).

(a) (15) Fill in the blanks in **torad()**:

```
sub torad {
   my $rad = shift;
   my $vl = shift;
   my @ary = ();
   if ($vl == 0) {@ary = (0);}
   else {
      while ($vl > 0) {
         $t = $vl % $rad;
         $vl = int($vl/$rad);
         --------------
      }
   }
   return ______________;
}
```

(b) (15) An instance method **printrad()** will print out the radix form of the stored number. For instance, if **$y** is storing 25, this function will print out 221. Fill in the blanks in the test code above (following the line printing **$x** and **$y**), so that the radix form of **$y** is printed out, and the blanks in the function itself below. For full credit, use no loops.

```
sub printrad {
   ------------------------
   ------------------------
   ------------------------
}
```

**4.** (15) In this problem you will write a class similar to Python's built-in **Queue** class, but for use with **pth**. The **get()** function returns with the head of the work queue; this occurs immediately if the queue is nonempty, but otherwise the call blocks until work is available. Fill in the gaps with at most a total of eight lines:

```
class pthqueue:
   def __init__(self,qid):
      self.qid = qid
      pth.newevent(qid)
      self.queue = []
   def get(self):
      (insert lines here)
   def put(self,work):
      (insert lines here)
```

**5.** (15) When they added OOP to Perl, the Perl development people made just two changes. First, they added the **bless** operation. Second, they changed the interpreter to allow functions in a package to be called via the object that had been blessed or the package itself, such as **$w1->printworker** and **Worker->newworker()** in our **Worker** example. However, show how to modify the calls to **newworker()** and **printworker()** in our example so as to avoid using either of those new Perl constructs; replace the existing four lines of code with four new lines of code. No changes are to be made to **Worker.pm** (other than removing the **bless**).

**Solutions:**

**1.**

```
x = z.pop(0)
z.insert(0,x)
sys.argv[4]
z[9:13]
```

**2.**

```
gb.scrn.addstr(gb.winrow,0,'kill this process?')
c = chr(gb.scrn.getch())
if c != 'y': return
pid = int(ln.split()[0])
os.kill(pid,9)
ln = ln[:15] + 'k' + ln[16:]  # can't change a tuple
gb.cmdoutlines[gb.startrow+gb.winrow)] = ln
gb.scrn.addstr(gb.winrow,0,ln,curses.A_BOLD)
```

**3.a**

```
unshift @ary,$k;
\@ary;
```

**3.b**

```
my $r = shift;
my $aryref = $r->{Digits};
print @$aryref;
...
$ry = tied $y;
$ry->printrad();
```

**4.**

```
def get():
   if self.queue == []:
      yield '','wait',self.qid
   return self.queue.pop(0)

def put():

   self.queue.append(work)
   if len(self.queue) == 1:
      yield '','set leave',self.qid
```

**5.**

```
$w1 = Worker::newworker('Worker','Cassius',12,50000)
$w2 = Worker::newworker('Worker','Penelope',5,90000)
...
Worker::printworker($w1)
Worker::printworker($w2)
```