PUNK0

PUNK ZERO or PUNK 0

PUNK0 is a game of function composition, each card's input is another card's output.

Each card has a function that takes a list of integers and returns a list of integers and each function is implemented in c, go, python and javascript.

The cards, especially the C cards, have a lot of bugs in them, they dont check for integer overflows and underflows, break down in various ways like malloc failing, but the idea of the card is clear.

The code is made more explicit, like not using slice operators, or using extra variables, or calling slices lists, but this way should be easier for beginners.

I hope you enjoy the game, and don't get discouraged if you lose track of the lists, just use a pen and paper to write it down on every step, thats what I do.

RULES

The starting list is:

[0, 0, 0, 0]

* Each player starts with 8 cards, the youngest player goes first and they can play any card they choose.

* Players can either play the same function in a different language or any function in the same language as the previous card.

* Players must play a matching card if they have one, otherwise they must draw one card from the deck.

* If a player is required to draw one or more cards as a result of a punk() card or because they don't have a matching card, they must play one of the cards they drew if it is a playable card.

* The first player to finish their cards wins the game.

* EASY MODE: you can play some cards as if they are the same function.
  + increment/decrement
  + rotate_left/rotate_right
  + punk0/punk1/punk2/punk3

PUNK CARDS

* punk0()

  The card will determine the next action based on the first element (index 0) of the list.

  + If the value of the first element is zero, the next player will skip their turn.

  + If the value is negative, the current player must play exactly N cards in the same turn, where N is the absolute value of the element. If they are unable to play N cards, they must draw from the deck.

  + If the value is positive, the next player must draw N cards from the deck, where N is the value of the element, unless they have a punk0 card, in which case they can forward the penalty to the next player.

* punk1() punk2() punk3()

  Same as punk0(), but using the appropriate index of the list, punk1 uses index 1 and so on.

```c
// filename: punk0.c
typedef struct list {
  size_t len;
  int32_t *data;
} list;

// returns a copy of the list and print
// what happens next:
//   [1,2,3,4]
// returns
//   [1,2,3,4]
list punk0(list x) {
  if (x.len >= 1) {
    int n = x.data[0];
    if (n == 0) {
      printf("next player skips\n");
    } else if (n < 0) {
      printf("play %d cards\n",-n);
    } else {
      printf("draw %d cards\n", n);
    }
  }
  // start with len=0 and allocate space
  // for x.len elements, 4 bytes each
  list r = {0, malloc(x.len * 4)};
  for (size_t i = 0; i < x.len; i++) {
    r.data[r.len++] = x.data[i];
  }

  return r;
}
```

```go
// filename: punk0.go


// returns a copy of the list and print
// what happens next:
//   [1,2,3,4]
// returns
//   [1,2,3,4]
func punk0(x []int) []int {
  if len(x) >= 1 {
    n := x[0]
    if n == 0 {
      fmt.Printf("next player skips\n")
    } else if n < 0 {
      fmt.Printf("play %d cards\n", -n)
    } else {
      fmt.Printf("draw %d cards\n", n)
    }
  }

  r := []int{}

  for i := range x {
    r = append(r, x[i])
  }

  return r
}
```

PK0 %----------> 06 <----------% PK0

```python
# filename: punk0.py
#
#
#
#
# returns a copy of the list and print
# what happens next:
#   [1,2,3,4]
# returns
#   [1,2,3,4]
def punk0(x):
  if len(x) >= 1:
    n = x[0]
    if n == 0:
      print('next player skips')
    elif n < 0:
      print(f'play {-n} cards')
    else:
      print(f'draw {n} cards')

  r = []

  for v in x:
    r.append(v)

  return r
```

```javascript
// filename: punk0.js


// returns a copy of the list and print
// what happens next:
//   [1,2,3,4]
// returns
//   [1,2,3,4]
function punk0(x) {
  if (x.length >= 1) {
    let n = x[0];
    if (n == 0) {
      console.log("next player skips");
    } else if (n < 0) {
      console.log(`play ${-n} cards`);
    } else {
      console.log(`draw ${n} cards`);
    }
  }

  let r = [];

  for (let v of x) {
    r.push(v);
  }

  return r;
}
```

```c
// filename: punk1.c
typedef struct list {
  size_t len;
  int32_t *data;
} list;

// returns a copy of the list and print
// what happens next:
//   [1,2,3,4]
// returns
//   [1,2,3,4]
list punk1(list x) {
  if (x.len >= 2) {
    int n = x.data[1];
    if (n == 0) {
      printf("next player skips\n");
    } else if (n < 0) {
      printf("play %d cards\n",-n);
    } else {
      printf("draw %d cards\n", n);
    }
  }
  // start with len=0 and allocate space
  // for x.len elements, 4 bytes each
  list r = {0, malloc(x.len * 4)};
  for (size_t i = 0; i < x.len; i++) {
    r.data[r.len++] = x.data[i];
  }

  return r;
}
```

```go
// filename: punk1.go


// returns a copy of the list and print
// what happens next:
//   [1,2,3,4]
// returns
//   [1,2,3,4]
func punk1(x []int) []int {
  if len(x) >= 2 {
    n := x[1]
    if n == 0 {
      fmt.Printf("next player skips\n")
    } else if n < 0 {
      fmt.Printf("play %d cards\n", -n)
    } else {
      fmt.Printf("draw %d cards\n", n)
    }
  }

  r := []int{}

  for i := range x {
    r = append(r, x[i])
  }

  return r
}
```

```python
# filename: punk1.py




# returns a copy of the list and print
# what happens next:
#   [1,2,3,4]
# returns
#   [1,2,3,4]
def punk1(x):
  if len(x) >= 2:
    n = x[1]
    if n == 0:
      print('next player skips')
    elif n < 0:
      print(f'play {-n} cards')
    else:
      print(f'draw {n} cards')

  r = []

  for v in x:
    r.append(v)

  return r
```

```javascript
// filename: punk1.js


// returns a copy of the list and print
// what happens next:
//   [1,2,3,4]
// returns
//   [1,2,3,4]
function punk1(x) {
  if (x.length >= 2) {
    let n = x[1];
    if (n == 0) {
      console.log("next player skips");
    } else if (n < 0) {
      console.log(`play ${-n} cards`);
    } else {
      console.log(`draw ${n} cards`);
    }
  }

  let r = [];

  for (let v of x) {
    r.push(v);
  }

  return r;
}
```

```c
// filename: punk2.c
typedef struct list {
  size_t len;
  int32_t *data;
} list;

// returns a copy of the list and print
// what happens next:
//   [1,2,3,4]
// returns
//   [1,2,3,4]
list punk2(list x) {
  if (x.len >= 2) {
    int n = x.data[2];
    if (n == 0) {
      printf("next player skips\n");
    } else if (n < 0) {
      printf("play %d cards\n",-n);
    } else {
      printf("draw %d cards\n", n);
    }
  }
  // start with len=0 and allocate space
  // for x.len elements, 4 bytes each
  list r = {0, malloc(x.len * 4)};
  for (size_t i = 0; i < x.len; i++) {
    r.data[r.len++] = x.data[i];
  }

  return r;
}
```

PK2 %----------> 13 <----------% PK2

```go
// filename: punk2.go


// returns a copy of the list and print
// what happens next:
//   [1,2,3,4]
// returns
//   [1,2,3,4]
func punk2(x []int) []int {
  if len(x) >= 2 {
    n := x[2]
    if n == 0 {
      fmt.Printf("next player skips\n")
    } else if n < 0 {
      fmt.Printf("play %d cards\n", -n)
    } else {
      fmt.Printf("draw %d cards\n", n)
    }
  }

  r := []int{}

  for i := range x {
    r = append(r, x[i])
  }

  return r
}
```

PK2 %----------> 14 <----------% PK2

```python
# filename: punk2.py




# returns a copy of the list and print
# what happens next:
#   [1,2,3,4]
# returns
#   [1,2,3,4]
def punk2(x):
  if len(x) >= 2:
    n = x[2]
    if n == 0:
      print('next player skips')
    elif n < 0:
      print(f'play {-n} cards')
    else:
      print(f'draw {n} cards')

  r = []

  for v in x:
    r.append(v)

  return r
```

```
// filename: punk2.js


// returns a copy of the list and print
// what happens next:
//   [1,2,3,4]
// returns
//   [1,2,3,4]
function punk2(x) {
  if (x.length >= 2) {
    let n = x[2];
    if (n == 0) {
      console.log("next player skips");
    } else if (n < 0) {
      console.log(`play ${-n} cards`);
    } else {
      console.log(`draw ${n} cards`);
    }
  }

  let r = [];

  for (let v of x) {
    r.push(v);
  }

  return r;
}
```

```c
// filename: punk3.c
typedef struct list {
  size_t len;
  int32_t *data;
} list;

// returns a copy of the list and print
// what happens next:
//   [1,2,3,4]
// returns
//   [1,2,3,4]
list punk3(list x) {
  if (x.len >= 3) {
    int n = x.data[3];
    if (n == 0) {
      printf("next player skips\n");
    } else if (n < 0) {
      printf("play %d cards\n",-n);
    } else {
      printf("draw %d cards\n", n);
    }
  }
  // start with len=0 and allocate space
  // for x.len elements, 4 bytes each
  list r = {0, malloc(x.len * 4)};
  for (size_t i = 0; i < x.len; i++) {
    r.data[r.len++] = x.data[i];
  }

  return r;
}
```

PK3 %----------> 17 <----------% PK3

```go
// filename: punk3.go


// returns a copy of the list and print
// what happens next:
//   [1,2,3,4]
// returns
//   [1,2,3,4]
func punk3(x []int) []int {
  if len(x) >= 3 {
    n := x[3]
    if n == 0 {
      fmt.Printf("next player skips\n")
    } else if n < 0 {
      fmt.Printf("play %d cards\n", -n)
    } else {
      fmt.Printf("draw %d cards\n", n)
    }
  }

  r := []int{}

  for i := range x {
    r = append(r, x[i])
  }

  return r
}
```

```python
# filename: punk3.py




# returns a copy of the list and print
# what happens next:
#   [1,2,3,4]
# returns
#   [1,2,3,4]
def punk3(x):
  if len(x) >= 3:
    n = x[3]
    if n == 0:
      print('next player skips')
    elif n < 0:
      print(f'play {-n} cards')
    else:
      print(f'draw {n} cards')

  r = []

  for v in x:
    r.append(v)

  return r
```

```
// filename: punk3.js


// returns a copy of the list and print
// what happens next:
//   [1,2,3,4]
// returns
//   [1,2,3,4]
function punk3(x) {
  if (x.length >= 3) {
    let n = x[3];
    if (n == 0) {
      console.log("next player skips");
    } else if (n < 0) {
      console.log(`play ${-n} cards`);
    } else {
      console.log(`draw ${n} cards`);
    }
  }

  let r = [];

  for (let v of x) {
    r.push(v);
  }

  return r;
}
```

```c
// filename: increment.c
typedef struct list {
  size_t len;
  int32_t *data;
} list;

// the player can specify
// which index to decrement
size_t INC_INDEX = 0;

// increment the INC_INDEX of a list,
// e.g. if INC_INDEX is set to 0:
//   [1,2,3,4]
// returns:
//   [2,2,3,4]
list increment(list x) {
  // start with len=0 and allocate space
  // for x.len elements, 4 bytes each
  list r = {0, malloc(x.len * 4)};

  for (size_t i = 0; i < x.len; i++) {
    int32_t v = x.data[i];
    if (i == INC_INDEX) {
      v++;
    }

    r.data[r.len++] = v;
  }

  return r;
}
```

```go
// filename: increment.go




// the player can specify
// which index to decrement
const INC_INDEX = 0
// increment the INC_INDEX of a list,
// e.g. if INC_INDEX is set to 0:
//   [1,2,3,4]
// returns:
//   [2,2,3,4]
func increment(x []int) []int {
  r := []int{}

  for i, v := range x {
    if i == INC_INDEX {
      v++
    }
    r = append(r, v)
  }

  return r
}
```

```
// filename: increment.js




// the player can specify
// which index to decrement
const INC_INDEX = 0
// increment the INC_INDEX of a list,
// e.g. if INC_INDEX is set to 0:
//   [1,2,3,4]
// returns:
//   [2,2,3,4]
function increment(x) {
  let r = [];

  for (let i = 0; i < x.length; i++) {
    let v = x[i]
    if (i == INC_INDEX) {
      v++;
    }
    r.push(v);
  }

  return r;
}
```

INC %----------> 23 <----------% INC

```python
# filename: increment.py




# the player can specify
# which index to decrement
INC_INDEX = 0
# increment the INC_INDEX of a list,
# e.g. if INC_INDEX is set to 0:
#   [1,2,3,4]
# returns:
#   [2,2,3,4]
def increment(x):
  r = []

  for i in range(len(x)):
    v = x[i]
    if i == INC_INDEX:
      v += 1
    r.append(v)

  return r
```

```c
// filename: increment.c
typedef struct list {
  size_t len;
  int32_t *data;
} list;

// the player can specify
// which index to decrement
size_t INC_INDEX = 0;

// increment the INC_INDEX of a list,
// e.g. if INC_INDEX is set to 0:
//   [1,2,3,4]
// returns:
//   [2,2,3,4]
list increment(list x) {
  // start with len=0 and allocate space
  // for x.len elements, 4 bytes each
  list r = {0, malloc(x.len * 4)};

  for (size_t i = 0; i < x.len; i++) {
    int32_t v = x.data[i];
    if (i == INC_INDEX) {
      v++;
    }

    r.data[r.len++] = v;
  }

  return r;
}
```

```
// filename: increment.go




// the player can specify
// which index to decrement
const INC_INDEX = 0
// increment the INC_INDEX of a list,
// e.g. if INC_INDEX is set to 0:
//   [1,2,3,4]
// returns:
//   [2,2,3,4]
func increment(x []int) []int {
  r := []int{}

  for i, v := range x {
    if i == INC_INDEX {
      v++
    }
    r = append(r, v)
  }

  return r
}
```

```
// filename: increment.js




// the player can specify
// which index to decrement
const INC_INDEX = 0
// increment the INC_INDEX of a list,
// e.g. if INC_INDEX is set to 0:
//   [1,2,3,4]
// returns:
//   [2,2,3,4]
function increment(x) {
  let r = [];

  for (let i = 0; i < x.length; i++) {
    let v = x[i]
    if (i == INC_INDEX) {
      v++;
    }
    r.push(v);
  }

  return r;
}
```

```
# filename: increment.py




# the player can specify
# which index to decrement
INC_INDEX = 0
# increment the INC_INDEX of a list,
# e.g. if INC_INDEX is set to 0:
#   [1,2,3,4]
# returns:
#   [2,2,3,4]
def increment(x):
  r = []

  for i in range(len(x)):
    v = x[i]
    if i == INC_INDEX:
      v += 1
    r.append(v)

  return r
```

```c
// filename: decrement.c
typedef struct list {
  size_t len;
  int32_t *data;
} list;

// the player can specify
// which index to decrement
size_t DEC_INDEX = 0;

// decrement the DEC_INDEX of a list,
// e.g. if DEC_INDEX is set to 0:
//   [1,2,3,4]
// returns:
//   [0,2,3,4]
list decrement(list x) {
  // start with len=0 and allocate space
  // for x.len elements, 4 bytes each
  list r = {0, malloc(x.len * 4)};

  for (size_t i = 0; i < x.len; i++) {
    int32_t v = x.data[i];
    if (i == DEC_INDEX) {
      v--;
    }

    r.data[r.len++] = v;
  }

  return r;
}
```

```go
// filename: decrement.go




// the player can specify
// which index to decrement
const DEC_INDEX = 0
// decrement the DEC_INDEX of a list,
// e.g. if DEC_INDEX is 0:
//   [1,2,3,4]
// returns:
//   [0,2,3,4]
func decrement(x []int) []int {
  r := []int{}

  for i, v := range x {
    if i == DEC_INDEX {
      v--
    }
    r = append(r, v)
  }

  return r
}
```

```
// filename: decrement.js




// the player can specify
// which index to decrement
const DEC_INDEX = 0
// decrement the DEC_INDEX of a list,
// e.g. if DEC_INDEX is set to 0:
//   [1,2,3,4]
// returns:
//   [0,2,3,4]
function decrement(x) {
  let r = [];

  for (let i = 0; i < x.length; i++) {
    let v = x[i]
    if (i == DEC_INDEX) {
      v--;
    }
    r.push(v);
  }

  return r;
}
```

```
# filename: decrement.py
#
#
#
#
#
# the player can specify
# which index to decrement
DEC_INDEX = 0
# decrement the DEC_INDEX of a list,
# e.g. if DEC_INDEX is set to 0:
#   [1,2,3,4]
# returns:
#   [0,2,3,4]
def decrement(x):
  r = []

  for i in range(len(x)):
    v = x[i]
    if i == DEC_INDEX:
      v -= 1
    r.append(v)

  return r
#
#
#
#
#
#
#
```

```c
// filename: decrement.c
typedef struct list {
  size_t len;
  int32_t *data;
} list;

// the player can specify
// which index to decrement
size_t DEC_INDEX = 0;

// decrement the DEC_INDEX of a list,
// e.g. if DEC_INDEX is set to 0:
//   [1,2,3,4]
// returns:
//   [0,2,3,4]
list decrement(list x) {
  // start with len=0 and allocate space
  // for x.len elements, 4 bytes each
  list r = {0, malloc(x.len * 4)};

  for (size_t i = 0; i < x.len; i++) {
    int32_t v = x.data[i];
    if (i == DEC_INDEX) {
      v--;
    }

    r.data[r.len++] = v;
  }

  return r;
}
```

DEC %----------> 33 <----------% DEC

```go
// filename: decrement.go




// the player can specify
// which index to decrement
const DEC_INDEX = 0
// decrement the DEC_INDEX of a list,
// e.g. if DEC_INDEX is 0:
//   [1,2,3,4]
// returns:
//   [0,2,3,4]
func decrement(x []int) []int {
  r := []int{}

  for i, v := range x {
    if i == DEC_INDEX {
      v--
    }
    r = append(r, v)
  }

  return r
}
```

```
// filename: decrement.js



// the player can specify
// which index to decrement
const DEC_INDEX = 0
// decrement the DEC_INDEX of a list,
// e.g. if DEC_INDEX is set to 0:
//   [1,2,3,4]
// returns:
//   [0,2,3,4]
function decrement(x) {
  let r = [];

  for (let i = 0; i < x.length; i++) {
    let v = x[i]
    if (i == DEC_INDEX) {
      v--;
    }
    r.push(v);
  }

  return r;
}
```

```
# filename: decrement.py
#
#
#
#
#
# the player can specify
# which index to decrement
DEC_INDEX = 0
# decrement the DEC_INDEX of a list,
# e.g. if DEC_INDEX is set to 0:
#   [1,2,3,4]
# returns:
#   [0,2,3,4]
def decrement(x):
  r = []

  for i in range(len(x)):
    v = x[i]
    if i == DEC_INDEX:
      v -= 1
    r.append(v)

  return r
#
#
#
#
#
#
#
#
```

```c
// filename: rotate_left.c
typedef struct list {
  size_t len;
  int32_t *data;
} list;

// rotate the input list to the left
//   [1,2,3,4]
// returns:
//   [2,3,4,1]
list rotate_left(list x) {
  // start with len=0 and allocate space
  // for x.len elements, 4 bytes each
  list r = {0, malloc(x.len * 4)};

  for (size_t i = 0; i < x.len; i++) {
    // go to the second element
    // then wrap around
    // example if x.len is 4:
    // (0 + 1) % 4 = 1
    // (1 + 1) % 4 = 2
    // (2 + 1) % 4 = 3
    // (3 + 1) % 4 = 0
    size_t idx = (i + 1) % x.len;
    int32_t v = x.data[idx];
    r.data[r.len++] = v;
  }

  return r;
}
```

```go
// filename: rotate_left.go




// rotate the input list to the left
//   [1,2,3,4]
// returns:
//   [2,3,4,1]
func rotate_left(x []int) []int {
  r := []int{}

  for i := 0; i < len(x); i++ {
    // go to the second element
    // then wrap around
    // example if x.len is 4:
    // (0 + 1) % 4 = 1
    // (1 + 1) % 4 = 2
    // (2 + 1) % 4 = 3
    // (3 + 1) % 4 = 0
    idx := (i + 1) % len(x)
    v := x[idx]
    r = append(r, v)
  }

  return r
}
```

```javascript
// filename: rotate_left.js




// rotate the input list to the left
//   [1,2,3,4]
// returns:
//   [2,3,4,1]
function rotate_left(x) {
  let r = [];
  for (let i = 0; i < x.length; i++) {
    // go to the second element
    // then wrap around
    // example if x.len is 4:
    // (0 + 1) % 4 = 1
    // (1 + 1) % 4 = 2
    // (2 + 1) % 4 = 3
    // (3 + 1) % 4 = 0
    let len = x.length;
    let idx = (i + 1) % len;
    let v = x[idx];
    r.push(v);
  }

  return r;
}
```

```python
# filename: rotate_left.py



# rotate the input list to the left
#   [1,2,3,4]
# returns:
#   [2,3,4,1]
def rotate_left(x):
  r = []

  i = 0
  for v in x:
    # go to the second element
    # then wrap around
    # example if x.len is 4:
    # (0 + 1) % 4 = 1
    # (1 + 1) % 4 = 2
    # (2 + 1) % 4 = 3
    # (3 + 1) % 4 = 0
    idx = (i + 1) % len(x)
    v = x[idx]
    r.append(v)
    i += 1

  return r
```

```c
// filename: rotate_left.c
typedef struct list {
  size_t len;
  int32_t *data;
} list;

// rotate the input list to the left
//   [1,2,3,4]
// returns:
//   [2,3,4,1]
list rotate_left(list x) {
  // start with len=0 and allocate space
  // for x.len elements, 4 bytes each
  list r = {0, malloc(x.len * 4)};

  for (size_t i = 0; i < x.len; i++) {
    // go to the second element
    // then wrap around
    // example if x.len is 4:
    // (0 + 1) % 4 = 1
    // (1 + 1) % 4 = 2
    // (2 + 1) % 4 = 3
    // (3 + 1) % 4 = 0
    size_t idx = (i + 1) % x.len;
    int32_t v = x.data[idx];
    r.data[r.len++] = v;
  }

  return r;
}
```

```go
// filename: rotate_left.go




// rotate the input list to the left
//   [1,2,3,4]
// returns:
//   [2,3,4,1]
func rotate_left(x []int) []int {
  r := []int{}

  for i := 0; i < len(x); i++ {
    // go to the second element
    // then wrap around
    // example if x.len is 4:
    // (0 + 1) % 4 = 1
    // (1 + 1) % 4 = 2
    // (2 + 1) % 4 = 3
    // (3 + 1) % 4 = 0
    idx := (i + 1) % len(x)
    v := x[idx]
    r = append(r, v)
  }

  return r
}
```

```javascript
// filename: rotate_left.js




// rotate the input list to the left
//   [1,2,3,4]
// returns:
//   [2,3,4,1]
function rotate_left(x) {
  let r = [];
  for (let i = 0; i < x.length; i++) {
    // go to the second element
    // then wrap around
    // example if x.len is 4:
    // (0 + 1) % 4 = 1
    // (1 + 1) % 4 = 2
    // (2 + 1) % 4 = 3
    // (3 + 1) % 4 = 0
    let len = x.length;
    let idx = (i + 1) % len;
    let v = x[idx];
    r.push(v);
  }

  return r;
}
```

```python
# filename: rotate_left.py



# rotate the input list to the left
#   [1,2,3,4]
# returns:
#   [2,3,4,1]
def rotate_left(x):
  r = []

  i = 0
  for v in x:
    # go to the second element
    # then wrap around
    # example if x.len is 4:
    # (0 + 1) % 4 = 1
    # (1 + 1) % 4 = 2
    # (2 + 1) % 4 = 3
    # (3 + 1) % 4 = 0
    idx = (i + 1) % len(x)
    v = x[idx]
    r.append(v)
    i += 1

  return r
```

```c
// filename: rotate_right.c

typedef struct list {
  size_t len;
  int32_t *data;
} list;

// rotate the input list to the right
//   [1,2,3,4]
// returns:
//   [4,1,2,3]
list rotate_right(list x) {
  // start with len=0 and allocate space
  // for x.len elements, 4 bytes each
  list r = {0, malloc(x.len * 4)};

  for (size_t i = 0; i < x.len; i++) {
    // go to the last element and
    // then wrap around
    // example if x.len is 4:
    // (0 + 4 - 1) % 4 = 3
    // (1 + 4 - 1) % 4 = 0
    // (2 + 4 - 1) % 4 = 1
    // (3 + 4 - 1) % 4 = 2
    size_t idx = (i + x.len - 1) % x.len;
    int32_t v = x.data[idx];
    r.data[r.len++] = v;
  }
  return r;
}
```

```go
// filename: rotate_right.go




// rotate the input list to the right
//   [1,2,3,4]
// returns:
//   [4,1,2,3]
func rotate_right(x []int) []int {
  r := []int{}

  for i := 0; i < len(x); i++ {
    // go to the last element and
    // then wrap around
    // example if len is 4:
    // (0 + 4 - 1) % 4 = 3
    // (1 + 4 - 1) % 4 = 0
    // (2 + 4 - 1) % 4 = 1
    // (3 + 4 - 1) % 4 = 2
    idx := (i + len(x) - 1) % len(x)
    v := x[idx]
    r = append(r, v)
  }

  return r
}
```

```javascript
// filename: rotate_right.js



// rotate the input list to the right
//   [1,2,3,4]
// returns:
//   [4,1,2,3]
function rotate_right(x) {
  let r = [];

  for (let i = 0; i < x.length; i++) {
    // go to the last element and
    // then wrap around
    // example if len is 4:
    // (0 + 4 - 1) % 4 = 3
    // (1 + 4 - 1) % 4 = 0
    // (2 + 4 - 1) % 4 = 1
    // (3 + 4 - 1) % 4 = 2
    let len = x.length;
    let idx = (i + len - 1) % len;
    let v = x[idx];
    r.push(v);
  }

  return r;
}
```

```python
# filename: rotate_right.py



# rotate the input list to the right
#   [1,2,3,4]
# returns:
#   [4,3,2,1]
def rotate_right(x):
  r = []

  i = 0
  for v in x:
    # go to the last element and
    # then wrap around
    # example if len is 4:
    # (0 + 4 - 1) % 4 = 3
    # (1 + 4 - 1) % 4 = 0
    # (2 + 4 - 1) % 4 = 1
    # (3 + 4 - 1) % 4 = 2
    idx = (i + len(x) - 1) % len(x)
    v = x[idx]
    r.append(v)
    i += 1

  return r
```

```c
// filename: rotate_right.c

typedef struct list {
  size_t len;
  int32_t *data;
} list;

// rotate the input list to the right
//   [1,2,3,4]
// returns:
//   [4,1,2,3]
list rotate_right(list x) {
  // start with len=0 and allocate space
  // for x.len elements, 4 bytes each
  list r = {0, malloc(x.len * 4)};

  for (size_t i = 0; i < x.len; i++) {
    // go to the last element and
    // then wrap around
    // example if x.len is 4:
    // (0 + 4 - 1) % 4 = 3
    // (1 + 4 - 1) % 4 = 0
    // (2 + 4 - 1) % 4 = 1
    // (3 + 4 - 1) % 4 = 2
    size_t idx = (i + x.len - 1) % x.len;
    int32_t v = x.data[idx];
    r.data[r.len++] = v;
  }
  return r;
}
```

```go
// filename: rotate_right.go




// rotate the input list to the right
//   [1,2,3,4]
// returns:
//   [4,1,2,3]
func rotate_right(x []int) []int {
  r := []int{}

  for i := 0; i < len(x); i++ {
    // go to the last element and
    // then wrap around
    // example if len is 4:
    // (0 + 4 - 1) % 4 = 3
    // (1 + 4 - 1) % 4 = 0
    // (2 + 4 - 1) % 4 = 1
    // (3 + 4 - 1) % 4 = 2
    idx := (i + len(x) - 1) % len(x)
    v := x[idx]
    r = append(r, v)
  }

  return r
}
```

```javascript
// filename: rotate_right.js



// rotate the input list to the right
//   [1,2,3,4]
// returns:
//   [4,1,2,3]
function rotate_right(x) {
  let r = [];

  for (let i = 0; i < x.length; i++) {
    // go to the last element and
    // then wrap around
    // example if len is 4:
    // (0 + 4 - 1) % 4 = 3
    // (1 + 4 - 1) % 4 = 0
    // (2 + 4 - 1) % 4 = 1
    // (3 + 4 - 1) % 4 = 2
    let len = x.length;
    let idx = (i + len - 1) % len;
    let v = x[idx];
    r.push(v);
  }

  return r;
}
```

```python
# filename: rotate_right.py



# rotate the input list to the right
#   [1,2,3,4]
# returns:
#   [4,3,2,1]
def rotate_right(x):
  r = []

  i = 0
  for v in x:
    # go to the last element and
    # then wrap around
    # example if len is 4:
    # (0 + 4 - 1) % 4 = 3
    # (1 + 4 - 1) % 4 = 0
    # (2 + 4 - 1) % 4 = 1
    # (3 + 4 - 1) % 4 = 2
    idx = (i + len(x) - 1) % len(x)
    v = x[idx]
    r.append(v)
    i += 1

  return r
```

```c
// filename: reset.c

typedef struct list {
  size_t len;
  int32_t *data;
} list;

// Note: the reset() card can be played
// on top of any card.

// create a new list
// returns:
//   [0,0,0,0]
list reset() {
  // start with len=0 and allocate space
  // for 4 elements, 4 bytes each
  list r = {0, malloc(4 * 4)};

  for (int i = 0; i < 4; i++) {
    r.data[r.len++] = 0;
    // same as:
    //   r.data[r.len] = 0
    //   r.len = r.len + 1
    // z++ means, use the value of z
    // and then add 1 to it
  }

  return r;
}
```

```
// filename: reset.go




// Note: the reset() card can be played
// on top of any card.

// create a new list
// returns:
//   [0,0,0,0]
func reset() []int {
  r := []int{}

  for i := 0; i < 4; i++ {
    r = append(r, 0)
  }

  return r
}
```

```
// filename: reset.js









// Note: the reset() card can be played
// on top of any card.

// create a new list
// returns:
//   [0,0,0,0]
function reset() {
  let r = [];

  for (let i = 0; i < 4; i++) {
    r.push(0);
  }

  return r;
}
```

```
# filename: reset.py




# Note: the reset() card can be played
# on top of any card.

# create a new list with the value
# 0,0,0,0
# returns:
def reset():
  r = []

  for i in range(4):
    r.append(0)

  return r


print(reset())
```