Recursion

Outline

What is Recursion?

What is Recursion?

A recursive function is a function that calls itself and meets the following conditions:

What is Recursion?

A recursive function is a function that calls itself and meets the following conditions:

- Has a base case

What is Recursion?

A recursive function is a function that calls itself and meets the following conditions:

- Has a base case
- Addresses subproblems that are smaller in some sense

What is Recursion?

A recursive function is a function that calls itself and meets the following conditions:

- Has a base case
- Addresses subproblems that are smaller in some sense
- Does not address subproblems that overlap

Examples

Examples

Recursive definition of the factorial function $n!$

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0 \text{, and} \\ 1 & \text{if } n = 0 \end{cases}$$

Examples

Recursive definition of the factorial function $n!$

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0 \text{, and} \\ 1 & \text{if } n = 0 \end{cases}$$

Implementation of $n!$ in Python

```
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)
```

Examples

Recursive definition of the factorial function $n!$

$$n! = \begin{cases} n(n-1)! & \text{if } n > 0, \text{ and} \\ 1 & \text{if } n = 0 \end{cases}$$

Implementation of $n!$ in Python

```
def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)
```

Call trace for `factorial(5)`

```
_factorial(5)
  _factorial(4)
    _factorial(3)
      _factorial(2)
        _factorial(1)
          _factorial(0)
            return 1
          return 1 * 1 = 1
        return 2 * 1 = 2
      return 3 * 2 = 6
    return 4 * 6 = 24
  return 5 * 24 = 120
```

Examples

Examples

Program: `factorial.py`

Examples

Program: `factorial.py`

- Command-line input: n (int)

Examples

Program: `factorial.py`

- Command-line input: n (int)
- Standard output: $n!$

Examples

Program: `factorial.py`

- Command-line input: n (int)
- Standard output: $n!$

```
>_ ~/workspace/ipp/programs

$ python3 factorial.py 0
1
$ python3 factorial.py 5
120
```

Examples

Examples

factorial.py

```python
import stdio
import sys

def main():
    n = int(sys.argv[1])
    stdio.writeln(_factorial(n))

def _factorial(n):
    if n == 0:
        return 1
    return n * _factorial(n - 1)

if __name__ == '__main__':
    main()
```

Examples

Examples

Recursive definition of Euclid's algorithm for computing the greatest common divisor (gcd) of p and q

$$\mathrm{gcd}(p, q) = \begin{cases} \mathrm{gcd}(q, p \bmod q) & \text{if } q \neq 0, \text{ and} \\ p & \text{if } q = 0 \end{cases}$$

Examples

Recursive definition of Euclid's algorithm for computing the greatest common divisor (gcd) of p and q

$$\mathrm{gcd}(p, q) = \begin{cases} \mathrm{gcd}(q, p \bmod q) & \text{if } q \neq 0 \text{, and} \\ p & \text{if } q = 0 \end{cases}$$

Implementation of $\mathrm{gcd}(p, q)$ in Python

```
def _gcd(p, q):
    if q == 0:
        return p
    return _gcd(q, p % q)
```

Examples

Recursive definition of Euclid's algorithm for computing the greatest common divisor (gcd) of p and q

$$\mathrm{gcd}(p, q) = \begin{cases} \mathrm{gcd}(q, p \bmod q) & \text{if } q \neq 0 \text{, and} \\ p & \text{if } q = 0 \end{cases}$$

Implementation of $\mathrm{gcd}(p, q)$ in Python

```python
def _gcd(p, q):
    if q == 0:
        return p
    return _gcd(q, p % q)
```

Call trace for `_gcd(1440, 408)`

```
_gcd(1440, 408)
  _gcd(408, 216)
    _gcd(216, 192)
      _gcd(192, 24)
        _gcd(24, 0)
          return 24
        return 24
      return 24
    return 24
  return 24
```

Examples

Examples

Program: `euclid.py`

Examples

Program: `euclid.py`

- Command-line input: p (int) and q (int)

Examples

Program: `euclid.py`

- Command-line input: p (int) and q (int)
- Standard output: $\gcd(p, q)$

Examples

Program: `euclid.py`

- Command-line input: p (int) and q (int)
- Standard output: $\gcd(p, q)$

```
>_ ~/workspace/ipp/programs

$ python3 euclid.py 1440 408
24
$ python3 euclid.py 314159 271828
1
```

Examples

Examples

euclid.py

```
def main():
    p = int(sys.argv[1])
    q = int(sys.argv[2])
    stdio.writeln(_gcd(p, q))

def _gcd(p, q):
    if q == 0:
        return p
    return _gcd(q, p % q)

if __name__ == '__main__':
    main()
```

Examples

Examples

Program: `towersofhanoi.py`

Examples

Program: `towersofhanoi.py`

- Command-line argument: n (int)

Examples

Program: `towersofhanoi.py`

- Command-line argument: n (int)
- Standard output: instructions to move n Towers of Hanoi disks to the left

Examples

Program: `towersofhanoi.py`

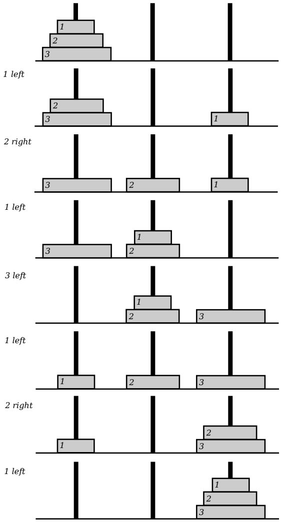
- Command-line argument: *n* (int)
- Standard output: instructions to move *n* Towers of Hanoi disks to the left

```
>_ ~/workspace/ipp/programs

$ python3 towersofhanoi.py 1
1 left
$ python3 towersofhanoi.py 2
1 right
2 left
1 right
$ python3 towersofhanoi.py 3
1 left
2 right
1 left
3 left
1 left
2 right
1 left
```

Examples

Examples



Examples

Examples

towersofhanoi.py

```
import stdio
import sys

def main():
    n = int(sys.argv[1])
    _moves(n, True)

def _moves(n, left):
    if n == 0:
        return
    _moves(n - 1, not left)
    if left:
        stdio.writeln(str(n) + ' left')
    else:
        stdio.writeln(str(n) + ' right')
    _moves(n - 1, not left)

if __name__ == '__main__':
    main()
```

Examples

Examples

Call trace for `_moves(3, True)`

```
_moves(3, True)
  _moves(2, False)
    _moves(1, True)
      _moves(0, False)
      1 left
      _moves(0, False)
    2 right
    _moves(1, True)
      _moves(0, False)
      1 left
      _moves(0, False)
  3 left
  _moves(2, False)
    _moves(1, True)
      _moves(0, False)
      1 left
      _moves(0, False)
    2 right
    _moves(1, True)
      _moves(0, False)
      1 left
      _moves(0, False)
```

Examples

Examples

Program: `htree.py`

Examples

Program: `htree.py`

- Command-line input: n (int)

Examples

Program: `htree.py`

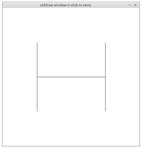
- Command-line input: n (int)
- Standard draw output: a level n H-tree centered at $(0.5, 0.5)$ with lines of length 0.5

Examples

Program: `htree.py`

- Command-line input: n (int)
- Standard draw output: a level n H-tree centered at $(0.5, 0.5)$ with lines of length 0.5

```
>_ ~/workspace/ipp/programs
$ python3 htree.py 1
```

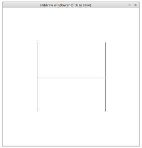


Examples

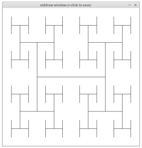
Program: `htree.py`

- Command-line input: n (int)
- Standard draw output: a level n H-tree centered at $(0.5, 0.5)$ with lines of length 0.5

```
>_ ~/workspace/ipp/programs
$ python3 htree.py 1
```




```
>_ ~/workspace/ipp/programs
$ python3 htree.py 3
```



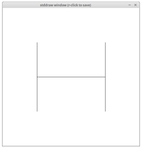
Examples

Program: `htree.py`

- Command-line input: n (int)
- Standard draw output: a level n H-tree centered at $(0.5, 0.5)$ with lines of length 0.5

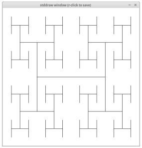
```
>_ ~/workspace/ipp/programs

$ python3 htree.py 1
```



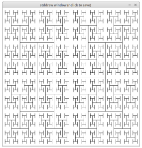
```
>_ ~/workspace/ipp/programs

$ python3 htree.py 3
```



```
>_ ~/workspace/ipp/programs

$ python3 htree.py 5
```



Examples

Examples

htree.py

```
import stddraw
import sys

def main():
    n = int(sys.argv[1])
    stddraw.setPenRadius(0.0)
    _draw(n, 0.5, 0.5, 0.5)
    stddraw.show()

def _draw(n, lineLength, x, y):
    if n == 0:
        return
    x0 = x - lineLength / 2
    x1 = x + lineLength / 2
    y0 = y - lineLength / 2
    y1 = y + lineLength / 2
    stddraw.line(x0, y, x1, y)
    stddraw.line(x0, y0, x0, y1)
    stddraw.line(x1, y0, x1, y1)
    _draw(n - 1, lineLength / 2, x0, y0)
    _draw(n - 1, lineLength / 2, x0, y1)
    _draw(n - 1, lineLength / 2, x1, y0)
    _draw(n - 1, lineLength / 2, x1, y1)

if __name__ == '__main__':
    main()
```

Examples

Examples

Program: `fibonacci.py`

Examples

Program: `fibonacci.py`

- Command-line input: n (int)

Examples

Program: `fibonacci.py`

- Command-line input: n (int)
- Standard output: nth Fibonacci number

Examples

Program: `fibonacci.py`

- Command-line input: n (int)
- Standard output: nth Fibonacci number

```
>_ ~/workspace/ipp/programs

$ python3 fibonacci.py 0
1
$ python3 fibonacci.py 1
1
$ python3 fibonacci.py 2
1
$ python3 fibonacci.py 3
2
$ python3 fibonacci.py 10
55
```

Examples

Examples

fibonacci.py

```python
import stdio
import sys

def main():
    n = int(sys.argv[1])
    stdio.writeln(_fibonacci(n))

def _fibonacci(n):
    if n < 2:
        return n
    return _fibonacci(n - 1) + _fibonacci(n - 2)

if __name__ == '__main__':
    main()
```

Pitfalls

Pitfalls

Missing base case

```
def _factorial(n):
    return n * _factorial(n - 1)
```

Pitfalls

Missing base case

```python
def _factorial(n):
    return n * _factorial(n - 1)
```

Recursion does not address smaller subproblems

```python
def _factorial(n):
    if n == 1:
        return 1
    return n * _factorial(n)
```

Pitfalls

Missing base case

```
def _factorial(n):
    return n * _factorial(n - 1)
```

Recursion does not address smaller subproblems

```
def _factorial(n):
    if n == 1:
        return 1
    return n * _factorial(n)
```

Recursion addresses overlapping subproblems

```
def _fibonacci(n):
    if n < 2:
        return n
    return _fibonacci(n - 1) + _fibonacci(n - 2)
```

Pitfalls

Missing base case

```
def _factorial(n):
    return n * _factorial(n - 1)
```

Recursion does not address smaller subproblems

```
def _factorial(n):
    if n == 1:
        return 1
    return n * _factorial(n)
```

Recursion addresses overlapping subproblems

```
def _fibonacci(n):
    if n < 2:
        return n
    return _fibonacci(n - 1) + _fibonacci(n - 2)
```

A function calls itself an excessive number of times before reaching the base case