PDSP 2026, Lecture 12, 17 September 2026

- Recursive definition of factorial
- Add diagnostic `print()` to trace recursive calls

In [1]:
```
def fact(n):
    print("evaluating factorial of",n)
    if n == 0:
        return(1)
    else:
        return(n*fact(n-1))
```

In [2]:
```
fact(8)
```

```
evaluating factorial of 8
evaluating factorial of 7
evaluating factorial of 6
evaluating factorial of 5
evaluating factorial of 4
evaluating factorial of 3
evaluating factorial of 2
evaluating factorial of 1
evaluating factorial of 0
```

Out[2]: 40320

- A negative argument creates an unending sequence of recursive calls
- Python has a built in `recursion limit` to abort such cases
  - This limit may deny some legitimate computation -- for instance `factorial(1000)` will also abor

- We can fix the negative number case by making the base case check stronger

In [3]:
```
def fact(n):
    print("evaluating factorial of",n)
    if n <= 0:
        return(1)
    else:
        return(n*fact(n-1))
```

In [4]:
```
fact(-1)
```

```
evaluating factorial of -1
```

Out[4]: 1

In [5]:
```
fact(1.5)
```

```
evaluating factorial of 1.5
evaluating factorial of 0.5
evaluating factorial of -0.5
```

Out[5]: 0.75

- `fact(1.5)` returns $1.5 \times 0.5 \times 1$

- Can manually increase the recursion limit
- Maximum is $2^{31} - 1$, the largest `int` value that can be stored in 32 bits

In [6]:
```python
import sys
sys.setrecursionlimit(2**31-1)
```

- Comment out the print statement to reduce the length of the output

In [7]:
```python
def fact(n):
    # print("evaluating factorial of",n)
    if n <= 0:
        return(1)
    else:
        return(n*fact(n-1))
```

In [8]:
```python
fact(1000)
```

Out[8]:
```
402387260077093773543702433923003985719374864210714632543799910429938512
398629020592044208486969404800479988610197196058631666872994808558901323
829669944590997424504087073759918823627727188732519779505950995276120874
975462497043601418278094646496291056393887437886487337119181045825783647
849977012476632889835955735432513185323958463075557409114262417474349347
553428646576611667797396668820291207379143853719588249808126867838374559
731746136085379534524221586593201928090878297308431392844403281231558611
036976801357304216168747609675871348312025478589320767169132448426236131
412508780208000261683151027341827977704784635868170164365024153691398281
264810213092761244896359928705114964975419909342221566832572080821333186
116811553615836546984046708975602900950537616475847728421889679646244945
160765353408198901385442487984959953319101723355556602139450399736280750
137837615307127761926849034352625200015888535147331611702103968175921510
907788019393178114194545257223865541461062892187960223838971476088506276
862967146674697562911234082439208160153780889893964518263243671616762179
168909779911903754031274622289988005195444414282012187361745992642956581
746628302955570299024324153181617210465832036786906117260158783520751516
284225540265170483304226143974286933061690897968482590125458327168226458
066526769958652682272807075781391858178889652208164348344825993266043367
660176999612831860788386150279465955131156552036093988180612138558600301
435694527224206344631797460594682573103790084024432438465657245014402821
885252470935190620929023136493273497565513958720559654228749774011413346
962715422845862377387538230483865688976461927383814900140767310446640259
899490222221765904339901886018566526485061799702356193897017860040811889
729918311021171229845901641921068884387121855646124960798722908519296819
372388642614839657382291123125024186649353143970137428531926649875337218
940694281434118520158014123344828015051399694290153483077644569099073152
433278288269864602789864321139083506217095002597389863554277196742822248
757586765752344220207573630569498825087968928162753848863396909959826280
956121450994871701244516461260379029309120889086942028510640182154399457
156805941872748998094254742173582401063677404595741785160829230135358081
840096996372524230560855903700624271243416909004153690105933983835777939
410970027753472000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000
```

- Another well known function that is defined inductively is the Fibonacci sequence
  - $fib(0) = 0$
  - $fib(1) = 1$
  - For $n > 1$, $fib(n) = fib(n-1) + fib(n-2)$
- The recursive implementation follows the same structure as the inductive definition
- However `fib(n)` takes a very long time even for relatively small values of `n` --- we will explore this later

In [9]:
```
def fib(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fib(n-1) + fib(n-2)
```

In [10]:
```
fib(8)
```

Out[10]: 21

In [11]:
```
fib(20)
```

Out[11]: 6765

In [12]:
```
fib(100)
```

```
---------------------------------------------------------------------------
-
KeyboardInterrupt                         Traceback (most recent call las
t)
Cell In[12], line 1
----> 1 fib(100)

Cell In[9], line 6, in fib(n)
      2     if n == 0:
      3         return 0
      4     if n == 1:
      5         return 1
----> 6     return fib(n-1) + fib(n-2)

…

Cell In[9], line 6, in fib(n)
      2     if n == 0:
      3         return 0
      4     if n == 1:
      5         return 1
----> 6     return fib(n-1) + fib(n-2)

Cell In[9], line 6, in fib(n)
      2     if n == 0:
      3         return 0
      4     if n == 1:
      5         return 1
----> 6     return fib(n-1) + fib(n-2)

Cell In[9], line 1, in fib(n)
----> 1 def fib(n):
      2     if n == 0:
      3         return 0
      4     if n == 1:

KeyboardInterrupt:
```

- Our dictionary based implementation of lists from last time

In [13]:
```
def createlist():  # Equivalent of l = [] is l = createlist()
  return({})

def appendlist(l,x):
  if l == {}:
    l["value"] = x
    l["next"] = {}
    return

  node = l
  while node["next"] != {}:
    node = node["next"]

  node["next"]["value"] = x
  node["next"]["next"] = {}
  return

def insertlist(l,x):  # l.insert(0,x)
  if l == {}:
```

```
    l["value"] = x
    l["next"] = {}
    return

  newnode = {}
  newnode["value"] = l["value"]
  newnode["next"] = l["next"]
  l["value"] = x
  l["next"] = newnode
  return


def printlist(l):
  print("{",end="")

  if l == {}:
    print("}")
    return
  node = l

  print(node["value"],end="")
  while node["next"] != {}:
    node = node["next"]
    print(",",node["value"],end="")
  print("}")
  return
```

- A recursive defintion of append

In [14]:
```
def emptylist(l):
    return l == {}

def appendlistr(l,v):  # Recursive definition of append
    if emptylist(l):
        l["value"] = v
        l["next"] = {}
    else:
        appendlistr(l["next"],v)
```

In [15]:
```
newl = createlist()
for i in range(10):
    appendlistr(newl,i)
printlist(newl)
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

- Other recursive functions on lists: length and sum

In [16]:
```
def lengthlist(l):
    if emptylist(l):
        return 0
    else:
        return 1 + lengthlist(l["next"])
```

In [17]:
```
lengthlist(newl)
```

Out[17]:
```
10
```

In [18]:
```
def sumlist(l):
    if emptylist(l):
        return 0
    else:
        return l["value"] + sumlist(l["next"])
```

In [19]:
```
sumlist(newl)
```

Out[19]: 45

- `deletelist(l,v)` deletes the first occurrence of `v` in `l`. If `v` is not present in `l`, the list is left undisturbed
- Here is a recursive implementation of `deletelist`

In [20]:
```
def deletelist(l,v):
    if emptylist(l):
        return
    if emptylist(l["next"]):
        if l["value"] == v:
            del(l["value"])
            del(l["next"])
        return
    if l["value"] == v:
        l["value"] = l["next"]["value"]
        l["next"] = l["next"]["next"]
    else:
        deletelist(l["next"],v)
```

In [21]:
```
printlist(newl)
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

- Check that this works with some representative operatio

- Delete from middle

In [22]:
```
deletelist(newl,5)
```

In [23]:
```
printlist(newl)
```

```
{0, 1, 2, 3, 4, 6, 7, 8, 9}
```

- Delete first element

In [24]:
```
deletelist(newl,0)
```

In [25]:
```
printlist(newl)
```

```
{1, 2, 3, 4, 6, 7, 8, 9}
```

- Delete last element

In [26]:
```
deletelist(newl,9)
```

In [27]:
```
printlist(newl)
```

```
{1, 2, 3, 4, 6, 7, 8}
```