Unit 1

Build a search engine

- Find data
- Build an index
- Rank Pages

Build web crawler

- command "print"

  # : annotation

- motivation for Programming language.

  natural language. → ambiguity
  → verbosity

- Backus-Naur Form

  < Non-terminal > → replacement

  derivation
  non-terminal ⟶ terminal statement

- Python Grammar for Arithmetic Expressions

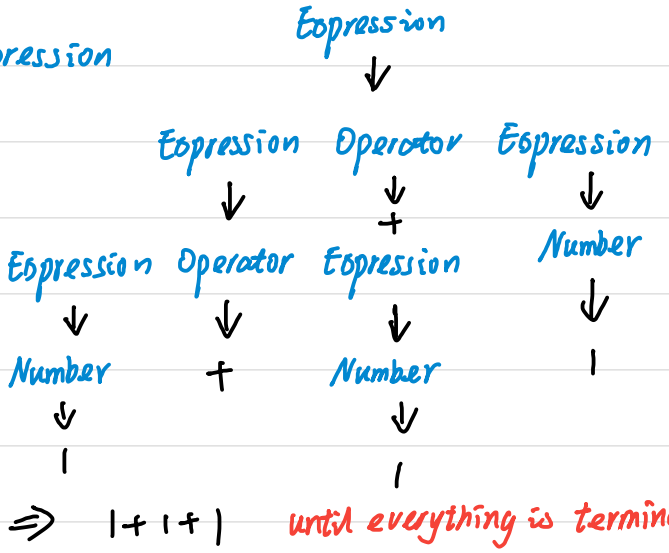
  Expression → Expression Operator Expression

  Expression → Number

  Operator → +, -, *, /

  Number → 0, 1, ...

  Expression → (Expression)

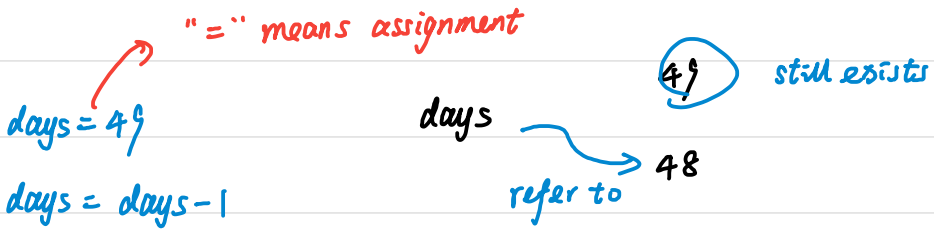


until everything is terminal

- Grace Hopper. COBOL. compiler

  arguably 据理

- Variables
  - Assignment statement.
  - variables can vary



  - < string > + < string > → concatenation

    < string > * number. eg. print '!' * 3 ⇒ !!!

Indexing Strings

eg. <string> [<expression>]

```
name = 'Dave'
print name[0]      } ⇒ 'D'

print name[-1]  ⇒ 'e'
```

Selecting Sub-Sequences

<string> [<expression>] → one character string

<string> [<start expression> : <stop expression>] starting from # start

eg. print word[3] s = 'Udacity' ending with # stop-1

print word[4:6] s[:] ⇔ s

print word[4:] s + s[0:0] ⇔ s

print word[:2] s[0:] ⇔ s

print word[:] s[:-1] ⇔ 'Udacit'

s[:3] + s[3:] ⇔ s # always true.

s = '' # empty string

s[0] → Error

s[12:12] → '' ✕ Error

Find

<string>.find(<string>)

① if found. starting position of the first character of <string>

② x found. return -1.

eg. s.find('') always returns with 0. because we don't need any string to find the empty string

s.find(s + ':::') + 1 always returns with 0.

<string>.find(<string>, <number>)

↓ starting point of <string>

eg. s[i:].find(t)

s.find(t)[:i] ⇔ ✕ s.find(t, i)

s[i:].find(t) + i → if t can be found in S, √

s[i:].find(t[i:]) t x exist in S, -1+i x equivalent to -1

✕ for the subsequent operator, s[0:] x cause error even though s is empty string. But s[0] will cause error.

✓ str(<number>) → <string> transform a number into a string

Unit 2 Procedure & Control


```
def <name>(<parameter>):
    <block>
```

✓ Making decisions

<number> <operator> <number> ⇒ Boolean value: either True or False

✓ if statement

```
if <Test Expression>:
    <Block>
else:
    <Block>
```

✓ or

<Expression> or <Expression>

✓ while

```
while <Test Expression>:
    <Block>
```

eg. factorial

```
def factorial(n):
    result = 1
    while n >= 1:
        result = result * n
        n = n - 1
    return result
```

✓ break

```
while <Test Expression>:
    <code>
    if <BreakTest>:
        Break
    <More code>
<After while>
```

Break → jump out of the loop

✓ s, t = t, s ⇒ swap the value of s and t

☆ def get_next_target(page):

```
    start_link = page.find('<a href=')
    if start_link == -1:
        return none, 0
    start_quote = page.find('"', start_link)
    end_quote = page.find('"', start_quote+1)
    url = page[start_quote+1, end_quote]
    return url, end_quote
```

get

```
def print_all_links(page):
    while True:
        url, end_pose = get_next_target(page)
        if url:
            print url   links.append(url)
            page = page[end_pos:]
        else:
            break
```

links=[]

return links

✓ assert <Expression>

Date1		Date2	
	before		True
	equal		True
	after		False

dateIsbefore(year1, month1, day1, year2, month2, day2)

Date1 before Date2 True

Date1 = Date2 False

Date1 after Date2 False

assert not dateIsbefore(year2, month2, day2, year1, month1, day1)

Date2 before Date1 True→False

Date2 = Date1 False → True

Date2 after Date1 False → True

Unit 3 Learning to crawl. Structured data

String List

<list> → [<Expression>, <Expression>, ...]

sequence of characters sequence of anything

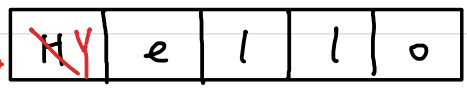
- Nested lists
- Mutation

eg. s = 'Hello'

no modifications s = 'Yello'

eg. p = ['H', 'e', 'l', 'l', 'o']

p[0] = 'Y'



eg. p = ['H', 'e', 'l', 'l', 'o']

q = p

p[0] = 'Y' also change q.

- List Operations

① <list>.append(<element>) add new element

stooges = ['Moe', 'Larry', 'Curly']

stooges.append('Shemp')

② <list> + <list>

[0,1] + [2,3] = [0,1,2,3]

③ len(<list>) ⇒ # of elements

↳ can also be applied to strings

/ for loop

for <name> in < list>:

< Block>

/ index

< list>. index (< value>)

eg. p = [0, 1, 2]

print p. index(2)

/ in

< value> in < list> ⇒ True/ False

< value> not in < list>

eg. print 2 in p

/ pop

< list>. pop( ) → element mutate the list by removing and returning its last element

/☆ def crawl_web(seed):

tocrawl = [seed]

crawled = []

while tocrawl:

page = tocrawl.pop(page)

if page not in crawled:

union(tocrawl, get_all_links(get_page(page)))

crawled.append(page)

return crawled

```
def union(p, q)
  for e in q:
    if e not in p:
      p.append(e)
```

p[0] p[1] p[2]

p[0] = p[0]+p[1] = 1

p[1] = p[0]+p[1]+p[2] = 2

p[2] = p[0]+p[1]+p[0]+p[1]+p[2] = 3

Unit 4 Responding to queries.

✓ Data Structure [[keyword1, [url11, url12, ...],
[keyword2, [url21, url22, ...],
...
]

✓ Add to Index

eg. index[]

```
def add_to_index(index, keyword, url):
    for entry in index:
        if keyword == entry[0]:
            append.entry[1](url)
            return
    append_index([keyword, [url]])
```

✓ split operation — built-in operation

<string>.split()

✓ Triple quote

quote = """ ... divide a long string into
... multiple lines
"""

✓ split_string function

```
eg. def split_string(source, splitlist)
        output[]
        atsplit = True
        for char in source:
            if char in splitlist:
                atsplit = True
            else:
                if atsplit
                    output.append(char)
```

```
        ! atsplit = False
      else
          output[-1] = output[-1] + char
    return output
```

```
def add_page_to_index(index, url, content)
    words = split.content()
    for word in words
        add_to_index(index, word, url)
def crawl_web(seed):
    tocrawl = [seed]
    crawled = []          } index = []
    while tocrawl:
        page = tocrawl.pop()
        if page not in crawled:
            content = get_page(page)
            add_page_to_index(index, page, content)
            union(crawled, get_all_links(content))
            crawled.append(page)
    return index
def lookup(index, keyword):
    for entry in index:
        if entry[0] == keywork:
            return entry[1]
    return []
```

Unit 5

How programs run

cost ↗ time
cost ↘ memory

```
time.clock()
```

```
def time_execution(code):
    start = time.clock()
    result = eval(code)
    run_time = time.clock() - start
    return result, run_time
```

exe. time

look up faster —— Hash Table

Define a Hash Function

<string> → [hash_string] → Number 0 ··· b-1

# of buckets, b

ord(<one-letter string>) → Number

chr(<number>) → <one-letter string>

chr(ord(a)) → a

Modulus Operator

<number> % <modulus> → <remainder>

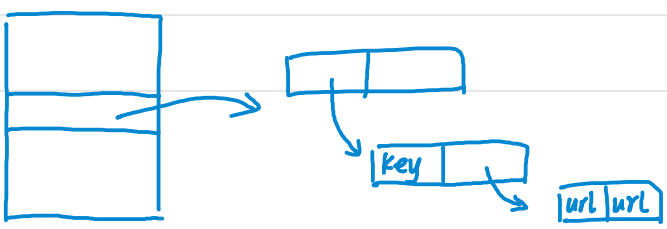
eg. 14 % 12 = 2

test hash function

```
eg. def test_hash_function(func, keys, size):
        results = [0] * size
        keys_used = []
        for w in keys:
            if w not in keys_used:
                hv = func(w, size)
                result[hv] = result[hv]+1
                keys_used.append(w)
        return results
```

Implement Hash Table

keyword → [hash_string] → buckets



[[[keyword, [url, url....]], ...], [[keyword, [url, url...]],...] , ...]

/ for e in <collection>

list or string

range(<start>, <stop>)

↳ [<start> , <start>+1, ..., <stop>-1]

eg. range(0, 10) → [0, 1, ... 9]

/ create n blank buckets

```
def make_hashtable(n):
    i=0
    table=[]
    while i<n:
        table.append([])
        i=i+1
    return table
```

→

```
def make-hashtable(n):
    table=[]
    for i in range(0, n):
        table.append([])
    return table
```

/

String	List	Dictionary	
'hello'	['alpha', 23]	{'hydrogen' : 1, 'helium' : 23	set of <key, value> pairs
sequence of characters	list of elements		
immutable	mutable	mutable → value of the key can be updated	
s[i].	p[i]=u	d[k]=v	

k ey

d[k] means value associated with key

Unit 6

How to have infinite power

/ recursive defination

eg. word → counter-word

word → anyword

Base case — a starting point

Recursive case

eg. factorial

factorial(n) = n * factorial(n-1)

factorial(0) = 1

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

eg. palindrome(s):

```
def is_palindrome(s):
    if s=='':
        return True
    else:
        if s[0]==s[-1]:
            return is_palindrome(s[1:-1])
        else:
            return False
```

eg. Fibonacci

fibonacci(0) = 0

fibonacci(1) = 1

fibonacci(n) = fibonacci(n-1) + fibonacci(n-2), n>1

eg. popularity(0, p) = 1

popularity(t, p) = $\sum_{f \in friend(p)}$ popularity(t-1, p)

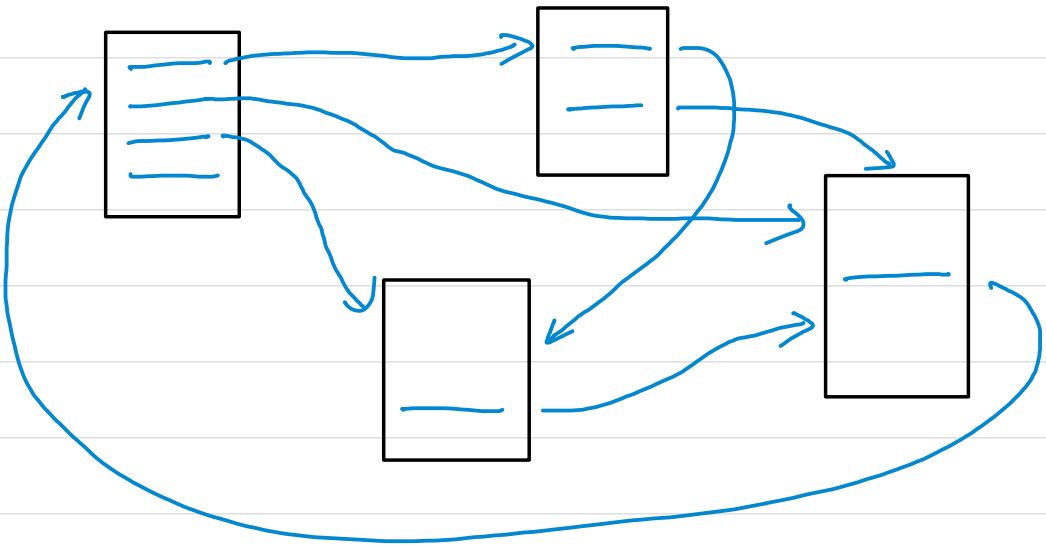
```
def popularity(t,p):
    if t==0:
        return 1
```

```
else:
    score = 0
    for f in friends(p):
        score = score + popularity(t-1, f)
    return score
```

eg. page ranking

$$rank(0, url) \to 1/N$$

$$rank(t, url) \to d \cdot \sum_{p \in inlinks[url]} rank(t-1, p) / outlinks[p] + (1-d) \cdot \frac{1}{N}$$



Dictionaries
- ranks — ranks at time t-1
- newranks — ranks at time t