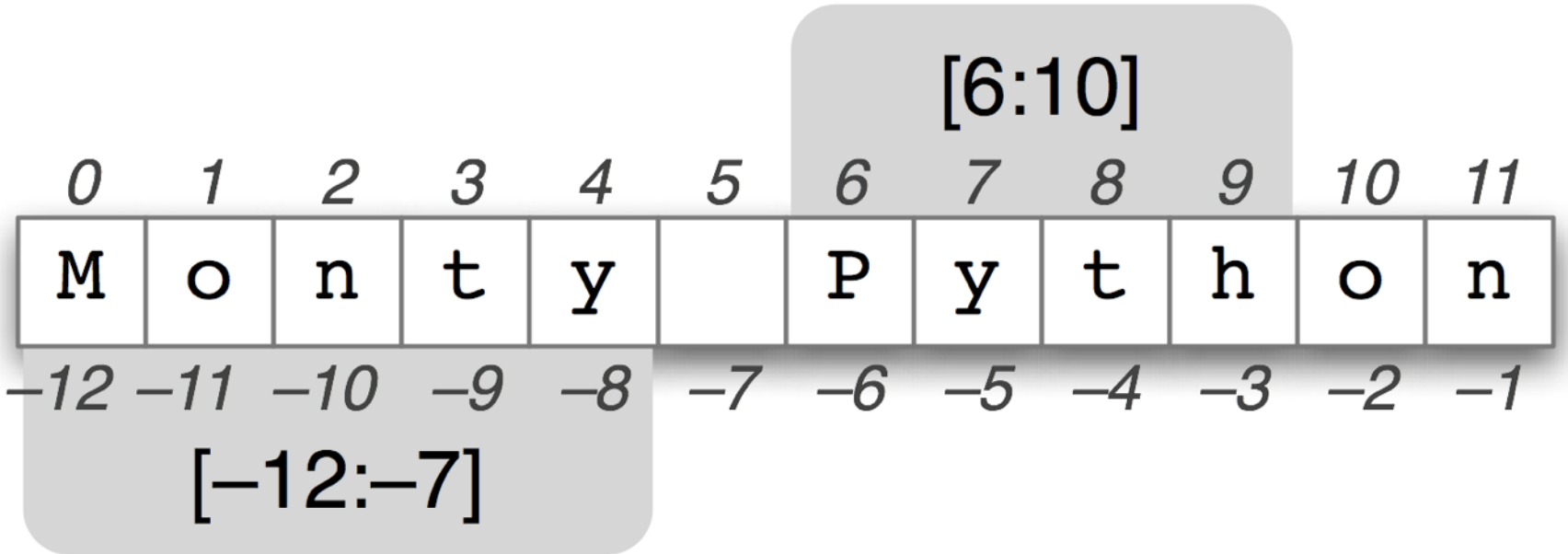
15-112
Fundamentals of Programming

Week 3 - Lecture 1:
Strings



January 26, 2016

Builtin Data Types

Python name	Description	Values
NoneType	absence of value	None
bool (boolean)	Boolean values	True, False
int (integer)	integer values	-2^{63} to $2^{63}-1$
long	large integer values	all integers
float	fractional values	e.g. 3.14
complex	complex values	e.g. 1+5j
str (string)	text	e.g. "Hello World!"
list	a list of values	e.g. [2, 5, "hello", "hi"]

...

String literals

String = A sequence (string) of characters.

String literals:

x = “#FeelTheBern” → string literal

x = ‘#FeelTheBern’ single-quotes

x = ‘‘‘#FeelTheBern’’’ triple single-quotes

x = “““#FeelTheBern””” triple double-quotes

What are the differences between these?

String literals

String = A sequence (string) of characters.

Single-quotes and double-quotes work similarly.

print(“hello world”)	hello world
print(‘hello world’)	hello world
print(“He said: “hello world”.”)	Syntax error
print(‘He said: “hello world”.’)	He said: “hello world”.
print(“He said: ‘hello world’.”)	He said: ‘hello world’.
print(“Hello World”)	Syntax error

String literals

String = A sequence (string) of characters.

Use triple quotes for multi-line strings.

```
print(“““hello
world”””)
```

hello
world

```
x = “‘#FeelTheBern
Hillary”’
```

```
print(x)
```

#FeelTheBern
Hillary

newline character

What value does x really store?

‘#FeelTheBern\nHillary’

String literals

String = A sequence (string) of characters.

\n newline \t tab

```
x = "#FeelTheBern\nHillary"

print(x)
```

```
#FeelTheBern
Hillary
```

```
x = "#FeelTheBern\tHillary"

print(x)
```

```
#FeelTheBern	Hillary
```

String literals

String = A sequence (string) of characters.

Escape characters: use \

print("The newline character is \n.")	The newline character is .
print("The newline character is \\n.")	The newline character is \n.
print("He said: \"hello world\".")	He said:"hello world".

String literals

String = A sequence (string) of characters.

Second functionality of \ : ignore newline

```
print(“‘#FeelTheBern
Hillary’”)
```

```
#FeelTheBern
Hillary
```

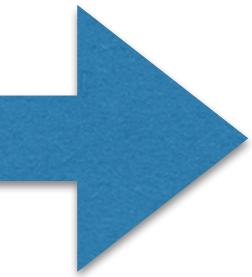
```
print(“‘#FeelTheBern \
Hillary’”)
```

```
#FeelTheBern Hillary
```

```
print(‘#FeelTheBern \
Hillary’)
```

```
#FeelTheBern Hillary
```

OUTLINE



String representation in memory

Built-in string operations

Built-in functions and constants related to strings

Built-in string methods

String formatting

String representation in memory

Every type of data in a computer is represented by numbers (binary numbers)

Each character in a string is a number.

```
print(ord("a"))        97
print(chr(97))         a
print(ord("b"))        98
print("a" < "b")       True
print("a" < "A")       False
print("A" < "a")       True
```

String representation in memory

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

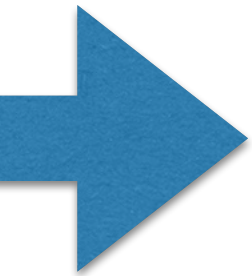
Example

Input: one character
Output: that character capitalized (if it is a letter).

```
def toUpperCaseLetter(c):
    if (("a" <= c) and (c <= "z")):
        return chr(ord(c) - (ord("a") - ord("A")))
    return c
```

OUTLINE

String representation in memory



Built-in string operations

Built-in functions and constants related to strings

Built-in string methods

String formatting

String gluing

Concatenation

```
print("Hello" + "World" + "!")
```
HelloWorld!

```
print("Hello"  "World"  "!")
```
HelloWorld!

```
s = "Hello"
```

```
print(s  "World"  "!")
```
ERROR

String gluing

Repetition

```
print("SPAM!!!" * 20)

print(20 * "SPAM!!!")

print(20 * "SPAM!!!" * 20)
```

String chopping

Indexing

G	o		T	a	r	t	a	n	s	!
0	1	2	3	4	5	6	7	8	9	10

s = “Go Tartans!”

print(s[0]) G

length = 11

print(s[5], s[length-1], s[3]) r ! T

expression that should evaluate to an integer

String chopping

Indexing

G	o		T	a	r	t	a	n	s	!
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

s = “Go Tartans!”

print(s[-1]) !

print(len(s)) 11

print(“Yabadabaduuu!”[5]) a

print(s[len(s)]) INDEX ERROR

String chopping

Slicing

G	o		T	a	r	t	a	n	s	!
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

s = “Go Tartans!”

print(s[3:len(s)]) Tartans!

print(s[0:len(s)]) Go Tartans!

print(s[3:]) Tartans!

print(s[:1]) G

print(s[:]) Go Tartans!

String chopping

Slicing

G	o		T	a	r	t	a	n	s	!
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

s = “Go Tartans!”

print(s[0:len(s):2])	G atn!
print(s[::])	Go Tartans!
print(s[len(s)-1:0:-1])	!snatraT o
print(s[len(s)-1:-1:-1])	range is empty, so it prints nothing
print(s[::-1])	!snatraT oG WEIRD!

Strings are immutable!!!!!

Slicing

G	o		T	a	r	t	a	n	s	!
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

s = “Go Tartans!”

s[3] = ‘t’ ERROR

s += “ haha”

print(s) Go Tartans! haha # Worked! Why?

s = s[:3] + ‘t’ + s[4:] effectively same as s[3] = “t”

print(s) Go tartans! haha

The in operator

The in operator returns True or False.

```
print("h" in "hello")          True
print("hell" in "hello")       True
print("ll" in "hello")         True
print("H" in "hello")          False
print("" in "hello")           True
print("k" not in "hello")      True
```

In a for loop, we also have in. Not the same as above.

```
for c in "112":
    print(c)
```

```
1
1
2
```

Example: getMonthName

Input: a number from 1 to 12
Output: first three letters of the corresponding month.

e.g. 1 returns "Jan", 2 returns "Feb", etc...

```
def getMonthName(monthNum):
    months = "JanFebMarAprMayJunJulAugSepOctNovDec"
    pos = (monthNum-1) * 3
    return months[pos:pos+3]
```

Example: indexOf

Input: a character c and a string s

Output: the index of the first occurence of c in s
(return -1 if c is not in s)

```
def indexOf(c, s):
    for index in range(len(s)):
        if (s[index] == c):
            return index
    return -1
```

Example: toUpperCase

Input: a string s

Output: a string with every letter in s capitalized

```
def toUpperCaseLetter(c):
    if (("a" <= c) and (c <= "z")):
        return chr(ord(c) - (ord("a") - ord("A")))
    return c


def toUpperCase(s):
    result = ""
    for c in s:
        result = result + toUpperCaseLetter(c)
    return result
```

Example: isPalindrome

Input: a string s

Output: True if s is a palindrome, False otherwise

Examples of palindromes: a, dad, hannah, civic

```
def isPalindrome(s):
    return s == s[::-1]
```

Example: isPalindrome

Input: a string s
Output: True if s is a palindrome, False otherwise

Examples of palindromes: a, dad, hannah, civic

```
def reverseString(s):
    return s[::-1]

def isPalindrome(s):
    return s == reverseString(s)
```

This strategy is not recommended.
You create a new string, which is not necessary.

Example: isPalindrome

Input: a string s

Output: True if s is a palindrome, False otherwise

Examples of palindromes: a, dad, hannah, civic

```
def isPalindrome2(s):
    mid = len(s)//2
    for i in range(mid):
        if (s[i] != s[-1-i]): return False
    return True
```

This is a good way of doing it.

Example: isPalindrome

Input: a string s

Output: True if s is a palindrome, False otherwise

Examples of palindromes: a, dad, hannah, civic

```
def isPalindrome2(s):
    mid = len(s)//2
    for i in range(mid):
        if (s[i] != s[len(s)-1-i]): return False
    return True
```

Most programming languages don't allow negative indices.

Example: isPalindrome

Input: a string s
Output: True if s is a palindrome, False otherwise

Examples of palindromes: a, dad, hannah, civic

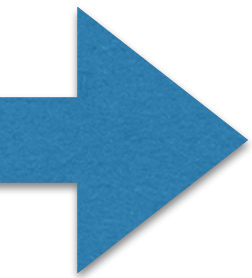
```
def isPalindrome3(s):
    while (len(s) > 1):
        if (s[0] != s[-1]): return False
        s = s[1:-1]
    return True
```

Even worse than the first one.

OUTLINE

String representation in memory

Built-in string operations



Built-in functions and constants related to strings

Built-in string methods

String formatting

Built-in functions

len( ), ord( ), chr( ), str( ), input( ), eval( )

```
print(len("hello\\\n\t"))        8
print(ord("A"))                  65
print(chr(85))                   U
print(str(85))                   85
userInput = input("How are you doing?")

print("2 + 3")                   2 + 3
print(eval("2 + 3"))             5
```

Built-in constants

```
import string

print(string.ascii_letters)

print(string.ascii_lowercase)

print(string.ascii_uppercase)

print(string.digits)

print(string.punctuation)

print(string.printable)

print(string.whitespace)

print("\n" in string.whitespace)
```

Example

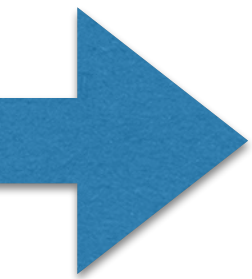
```
import string

def isLowercase(c):
    return (c in string.ascii_lowercase)
```

OUTLINE

String representation in memory

Built-in string operations

Built-in functions and constants related to strings



Built-in string methods

String formatting

Built-in string methods

Method: a function applied "directly" on an object/data

Example: there is a string method called upper( ),
it works like toUpperCase( ).

```
s = "hey you!"
```

```
print(upper(s))
```
ERROR: not used like a function.

```
print(s.upper())
```
HEY YOU!

```
s.upper( )
upper(s)
```
is kind of like
(if upper was a function)

Built-in string methods

Method: a function applied "directly" on an object/data

Example: there is a string method called count( ):

```
s = "hey hey you!"

print(s.count("hey"))      2
```

`s.count("hey")` is kind of like
`count(s, "hey")` (if count was a function)

Built-in string methods

String editing

```
print("hey you!".upper())
```
HEY YOU!

```
print("HEY YOU!".lower())
```
hey you!

```
s = "LeBron is the real number 23."
print(s.replace("LeBron", "MJ"))
```
MJ is the real number 23.

```
print("This is nice. Really nice.".replace("nice", "sweet"))
```
This is sweet. Really sweet.

```
print("This is nice. Really nice.".replace("nice", "sweet", 1))
```
This is sweet. Really nice.

```
print("   Strip removes leading and trailing whitespace   ".strip())
```

Built-in string methods

String editing

```
s = “HEY YOU!”
s.lower()
print(s)
```

HEY YOU!

```
s = s.lower()
print(s)
```

hey you!

Built-in string methods

Checking character types

print("112".isdigit())	True
print("abc112".isalnum())	True
print("abc".isalpha())	True
print("abc".islower())	True
print("ABC?!".isupper())	True
print("\n\t".isspace())	True

Built-in string methods

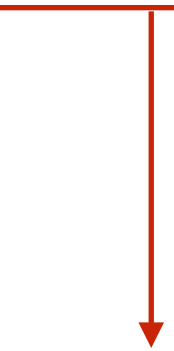
Substring search

print("This is a history test.".count("is"))	3
print("This is a history test.".startswith("This"))	True
print("This is a history test.".endswith("t."))	True
print("This is a history test.".find("is"))	2
print("This is a history test.".find("has"))	-1
print("This is a history test.".index("is"))	2
print("This is a history test.".index("has"))	CRASH

Built-in string methods

split and splitlines

```
names = "Alice,Bob,Charlie,David"

for name in names.split(","):
    print(name)
```



returns [Alice, Bob, Charlie, David]

```
Alice
Bob
Charlie
David
```

Built-in string methods

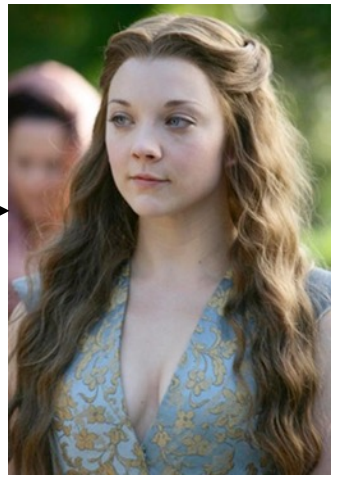
split and splitlines

```
s.splitlines()  ≈  s.split("\n")
```

quotes = """\
Dijkstra: Simplicity is prerequisite for reliability.
Knuth: If you optimize everything, you will always be unhappy.
Dijkstra: Perfecting oneself is as much unlearning as it is learning.
Knuth: Beware of bugs in the above code; I have only proved it correct, not tried it.
Dijkstra: Computer science is no more about computers than astronomy is about telescopes.
"""

```
for line in quotes.splitlines():
    if (line.startswith("Knuth")):
        print(line)
```

Example: Cryptography



"loru23n8uladjkfb!#@"

"I will cut your throat"

encryption

"loru23n8uladjkfb!#@"

"loru23n8uladjkfb!#@"

decryption

"I will cut your throat"

Example: Caesar shift

Encrypt messages by shifting each letter a certain number of places.

Example: shift by 3

a → d b → e c → f ... x → a y → b ...

A → D B → E ... X → A Y → B ...

(other symbols stay the same)

15112 Rocks my world → 15112 Urfvn pb zruog

Write methods to encrypt and decrypt messages.
(message and shift given as input)

Example: Caesar shift

```
def encrypt(message, shiftNum):
    result = “”
    for char in message:
        result += shift(char, shiftNum)
    return result


def shift(c, shiftNum):
    shiftNum %= 26
    if (not c.isalpha()):
        return c
    alph = string.ascii_lower if (c.islower())  else string.ascii_upper
    shifted_alph = alph[shiftNum:] + alph[:shiftNum]
    return shifted_alph[alph.find(c)]
```

Example: Caesar shift

```
def shift2(c, shiftNum):
    shiftNum %= 26
    if(('A' <= c) and (c <= 'Z')):
        if(ord(c) + shiftNum > ord('Z')):
            return chr(ord(c) + shiftNum - 26)
        else:
            return chr(ord(c) + shiftNum)

    elif(('a' <= c) and (c <= 'z')):
        if(ord(c) + shiftNum > ord('z')):
            return chr(ord(c) + shiftNum - 26)
        else:
            return chr(ord(c) + shiftNum)
    else:
        return c
```

Code repetition

Exercise: Rewrite avoiding the repetition

Tangent: Private-Key Cryptography

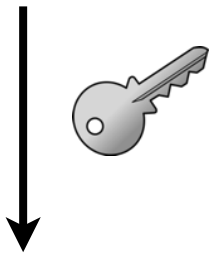
Cryptography before WWII

Tangent: Private-Key Cryptography

Cryptography before WWII

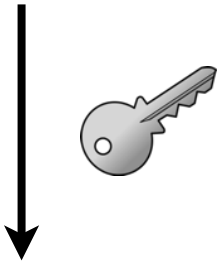
"#dfg%y@d2hSh2$&"

"I will cut your throat"



"#dfg%y@d2hSh2$&"

"#dfg%y@d2hSh2$&"



"I will cut your throat"

Tangent: Private-Key Cryptography

Cryptography before WWII

there must be a secure way of exchanging the key

Tangent: Public-Key Cryptography

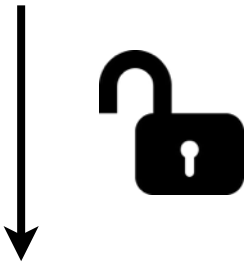
Cryptography <u>after</u> WWII

Tangent: Public-Key Cryptography

Cryptography after WWII

"#dfg%y@d2hSh2$&"

"I will cut your throat"



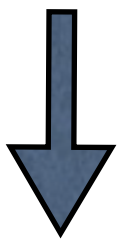
"#dfg%y@d2hSh2$&"

"#dfg%y@d2hSh2$&"

"I will cut your throat"

Tangent: The factoring problem

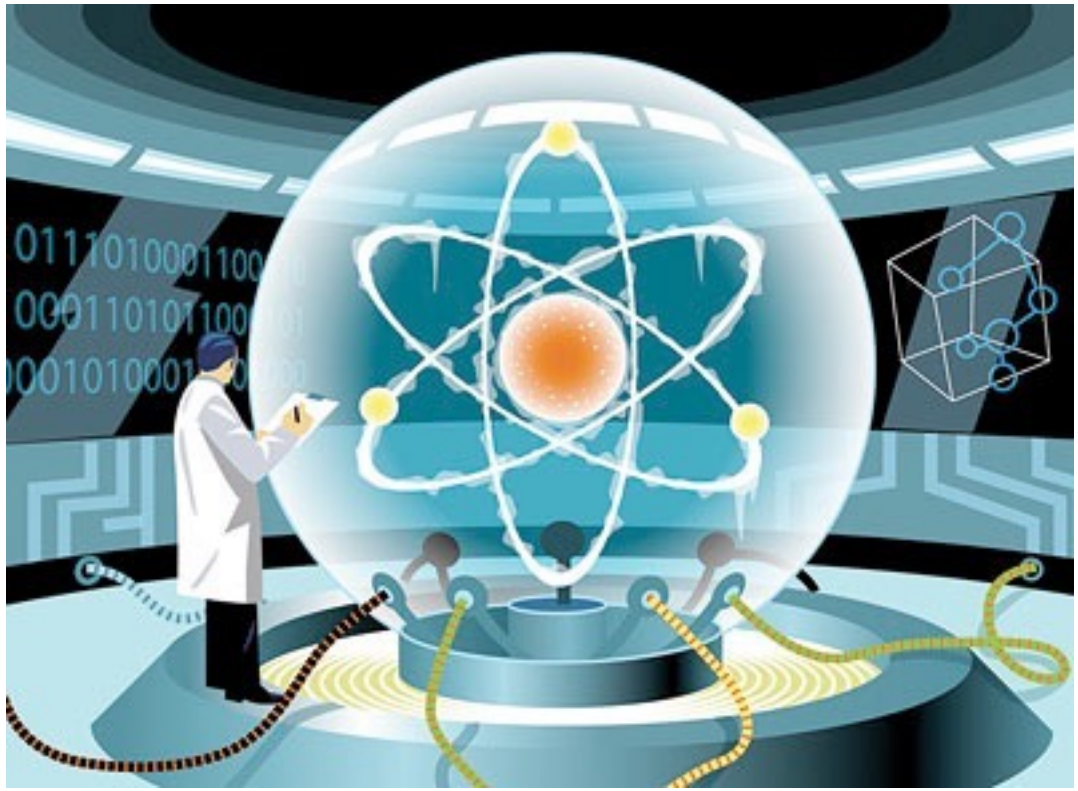
If there is an efficient program to solve the factoring problem



can break public-key crypto systems used over the internet

Fun fact: *Quantum computers* can factor large numbers efficiently!

Tangent: What is a quantum computer?



Information processing using quantum physics.

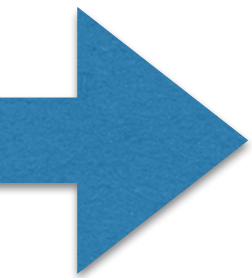
OUTLINE

String representation in memory

Built-in string operations

Built-in functions and constants related to strings

Built-in string methods



String formatting

On Thursday

Exercise for tonight:

Go over course notes.

Decrypt Caesar's message for you:

KvmGhirmKvsbMciQobKohqvUoasCtHvfcbsg