for is a sequential operator

```python
>>> o_ctr = 0
>>> for letter in 'hello':
...     if letter == 'o':
...         o_ctr += 1
```

strings are immutable sequences

```python
>>> s = 'jello, world'
>>> s[0] = 'h'
```

TypeError: 'str' object does not support item assignment

```python
>>> s = 'hello, world'
>>> s[1:] = 'h'
```

TypeError: 'str' object does not support item assignment

```python
>>> s = 'hello, world'
>>> s[1:] = 'h'
```

strings are immutable sequences

```python
>>> s = 'jello, world'
>>> s[0] = 'h'
```

TypeError: 'str' object does not support item assignment

```python
>>> s = 'hello, world'
>>> s[1:] = 'h'
```

strings are immutable sequences
while loop

indefinite iteration

most general type of loop

while <boolean expression>:
  statement1
  statement2
  ...
  statement

---

```python
p = 10
i = 1
ctr = 1  # initialize loop variable
while ctr <= p:  # check end condition
  i = i * 2  
  ctr += 1  # move loop var toward
print(i)  # the end condition
```

---

```python
p = 10
i = 1
ctr = 1  # loop var
for ctr in range(p):  # check
    i = i * 2
    ctr = ctr + 1  # advance
print(i)
```

---

```python
p = 10
i = 1
ctr = 1  # initialize loop variable
while ctr <= p:  # check
  i = i * 2
  ctr += 1  # move loop var toward
print(i)  # the end condition
```

---

```python
print('Enter numbers you wish to add, quit to end.
sum = 0
next = input('next: ')
while next != 'quit':
  sum += int(next)
  next = input('next: ')
print('Sum is:', sum)
```

---

**Computability (Automata) Theory:**

On computable numbers, with an application to the Entscheidungsproblem, A.M. Turing (1936)

definition of **computability** -- generally speaking,

-- memory/input/output -- assignment
-- execute code -- comparison
-- numbers -- go to
Computability (Automata) Theory:

definition of **computability** – generally speaking,

-- memory/input/output -- execute code
-- numbers -- comparison
-- assignment -- go to

Church-Turing Thesis:
Turing machine is a **universal machine**

A closer look:

• Python objects
• Python assignment

-- mental model of computational processes
-- for reading/writing more advanced code
-- for analyzing (preventing!) errors

What are Python’s primitive elements?

**OBJECT**

*type* – range of values and operations (attributes)
*value(s)*
*id* (memory location)

Examples of Python objects:

```python
>>> 99
99
>>> type(99)
<class 'int'>
>>> id(99)
2276160
```

```python
>>> 99.9
99.9
>>> type(99.9)
<class 'float'>
>>> id(99.9)
4298470336
```
what are the primitive elements?
-- object, with
-- type that determines range of values and operations (attributes)
-- value(s) of the object
-- memory location

Recall: Objects can be named
For example,
>>> x = 10

Variable assignment
<variable> = <expression>

>>> b = 20
>>> a = b + 1
>>> a
??

1. evaluate the expression on the rhs
2. associate name on lhs with resulting value

Variable assignment
<variable> = <expression>

>>> b = 20
>>> a = b + 1
>>> a
??

1. evaluate the expression on the rhs
2. associate name on lhs with resulting value
1. evaluate the expression on the rhs
2. associate name on lhs with resulting value

Let the variable on lhs refer to resulting object:

- allocate space in memory for the object
- search current namespace – if name on lhs is not there:
  - assign name on lhs to address of memory location
  - add it to the current namespace
- if name on lhs is there:
  - replace old reference with new reference (address)

```python
>>> b = 20
reference semantics:
>>> id(b)
4297645024
the name on the lhs is a reference (pointer)
>>> a = b + 1
>>> id(a)
4297645056
to the memory location of the data object
>>> id(a) == id(b)
False
>>> a
21
>>> b
?
```

1. evaluate the expression on the rhs
2. associate name on lhs with resulting value
>>> b = 20
>>> id(b)
4297645024

>>> a = b + 1
>>> id(a)
4297645056

>>> b = 30
>>> id(b)
4297645344

Evaluate rhs. If it is a variable name:
Let the variable on lhs refer to resulting object:
  – use address of rhs variable (no memory is allocated)
  – search current namespace – if name on lhs is not there:
    – assign name on lhs to address of memory location
    – add it to the current namespace
  – if name on lhs is there:
    – replace old reference with new reference (address)

>>> b = 20
>>> a = b
>>> b = 40
>>> a

>>> b = 20
>>> a = b + 1
>>> b = 40
CIS 210
>>> msg = 'hello'
>>> msg
'hello'
>>> greeting = msg
to the memory location of the data object
>>> id(msg)
4383118984
>>> id(greeting)
??

CIS 210
>>> msg = 'hello'
>>> msg
'hello'
>>> greeting = msg
to the memory location of the data object
>>> id(msg)
4383118984
>>> id(greeting)
4383118984
greeting “aliases” msg

CIS 210
>>> msg = 'goodbye'
>>> msg
'goodbye'
>>> id(msg)
4383119544
>>> id(greeting)
>>> greeting
4383118984

CIS 210
>>> msg = 'hello'
>>> msg = 'jello'
>>> class = 'CIS 210'
>>> class = 'CIS 211'

CIS 210
>>> msg = 'hello'
>>> msg = 'jello'
>>> class = 'CIS 210'
>>> class = 'CIS 211'
garbage collection

- Python interpreter
- check whether any variables are pointing at objects for which it has allocated memory
- if none are, the object is deleted and the memory is made available again

CIS 210
Assignment statements

Python is a dynamically typed language

```python
>>> a = 10
>>> type(a)
??
>>> b = a
>>> type(b)
??
```

CIS 210
Assignment statements

Python is a dynamically typed language

```python
>>> a = 10
>>> type(a)
??
>>> b = a
>>> type(b)
??
```

CIS 210
Assignment statements

- static typing
  - var a : int
  - a = 4
  - a = 'hello' X

CIS 210
Assignment statements

- Static, Dynamic – for example, scope, type
- static – can be determined by reading code (only)
- dynamic – scope/type is determined when code executes
CIS 210
Assignment statements

dynamic typing

greenCt = 1
for ctr in range(4):
    greenCt = greenCt + 1
print(greenCt)

??

Recall: we can combine objects in expressions, which are evaluated and return a value

For example,
>>> 99 + 10
??
>>> len('hello')
??
>>> str.center('****', 10)
??

Python is a strongly typed language

accumulator pattern

CIS 210
Assignment statements

dynamic typing

greenCt = 1
for ctr in range(4):
    greenCt += 1
print(greenCt)

??

Recall: we can combine objects in expressions, which are evaluated and return a value

For example,
>>> 99.9 + 100
??

Python is an extremely strongly typed language
def check(a: int, b: str) -> None:
    """(int, str) -> None
    Python 3.5 and higher
    ""
    print(a, b)
    return None

>>> check('hello', 99)  # needs interpreter support
hello 99

def check(a: int, b: str, c: string) -> None:
    ""
    print(a, b, c)
    return None

Traceback (most recent call last):
  File "/Users/kfreeman/Documents/cis210W18/projects-W18/check.py", line 1, in <module>
    def check(a:int, b:str, c:string):
NameError: name 'string' is not defined

>>> # closer look: Python objects/assignment
# attributes, type, id
# reference semantics
# garbage collection
# dynamic typing, strong typing

# binary representation of numbers
# binary ↔ decimal
-- hexadecimal representation of numbers
-- binary representation of characters

>>> twice
<function twice at 0x105b19b70>

>>> int(0x105b19b70)  # decimal
4390493040

>>> bin(0x105b19b70)  # binary
'0b100000101101100011001101110000'

>>> hex(4390493040)  # hexadecimal
'0x105b19b70'

Everything we do on a computer is an abstraction of a binary representation.

Binary sequences are used to represent all digital data.

Therefore, understanding binary representations of data is crucial for computer science.

Relevant topics include limitations of data types (e.g., floating point precision), computer organization, security, and many others.
Binary representation of numbers

Recall:

\[ 469_{10} = (4 \times 10^2) + (6 \times 10^1) + (9 \times 10^0) \]

Binary to decimal:

111010101_2 = (1 \times 2^8) + (1 \times 2^7) + (1 \times 2^6) + (0 \times 2^5) + (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)

• look at each bit in the binary number, \( b \).
• multiply it by the correct power of 2
• sum the results

"Decimal to decimal":

\[ \begin{align*}
469 & \div 10 \text{ q46 r9 } \rightarrow \text{ rightmost place} \\
46 & \div 10 \text{ q4 r6 } \rightarrow \text{ next rightmost place} \\
4 & \div 10 \text{ q0 r4 } \rightarrow \text{ next rightmost place} \\
469 & \\
\end{align*} \]

Decimal to binary:

\[ \begin{align*}
21 & \div 2 \text{ q10 r1 } \rightarrow \text{ rightmost bit} \\
10 & \div 2 \text{ q5 r0 } \rightarrow \text{ next bit} \\
5 & \div 2 \text{ q2 r1 } \rightarrow \text{ next bit} \\
2 & \div 2 \text{ q1 r0 } \rightarrow \text{ next bit} \\
1 & \div 2 \text{ q0 r1 } \rightarrow \text{ next bit} \\
10101 & \\
\end{align*} \]
CIS 210

Binary representation of numbers

Decimal to binary:
469 ÷ 2 q234 r1 → rightmost bit
234 ÷ 2 q117 r0 → next bit
117 ÷ 2 q58 r1 → next bit
58 ÷ 2 q29 r0 → next bit
29 ÷ 2 q14 r1 → next bit
14 ÷ 2 q7 r0 → next bit
7 ÷ 2 q3 r1 → next bit
3 ÷ 2 q1 r1 → next bit
1 ÷ 2 q0 r1 → leftmost bit

111010101

the initial value of quotient is the decimal number to be converted while quotient is greater than 0
divide the decimal number by 2
make the remainder the next digit to the left in the answer
update the quotient

Why hexadecimal?
Groupings of 4 bits are easier to read and write.
(1) (1101) (0101) binary

11c_{16} = (1 * 16^2) + (13 * 16^1) + (5 * 16^0)
CIS 210
Programming/Computer Science concepts.

Computational Problem Solving: designing, implementing, checking, revising algorithms/programs.

Good programming style: function docstrings (type contract; description including parameters, returned value, and side effects if any; examples of function use), well-named variables, use of whitespace between operators and sections of code, judicious use of inline comments (why not what).

Python is a programming language and Python is an interpreter (program)
Python Shell is a REPL (read-evaluate-print loop)
Python primitive elements: objects - value/attributes, type, memory location; memory management; garbage collection; immutable data type
Combining primitive elements: Expressions - expressions evaluate to a value; short circuit evaluation of boolean expressions; overloaded operators
Naming values: Variables/assignment - assignment statements are not expressions and do not return a value; namespaces – builtins and global/__.main__/._scope.

Functions are an executable data type; what happens when a function - method - is called:
Activation record/stack frame added to call stack for local namespace; return address
Call-by-assignment parameter passing
Functions always return a value (sometimes None)
Functions sometimes have side effects

Iterative algorithms; accumulator pattern; Monte Carlo algorithms

CIS 210
Python toolkit so far
numeric data types (int, float) and operations (e.g., +, **, round, abs)
string data type and operations (e.g., +, len, count, find)
Boolean data type and relational/Boolean operations (e.g., <, and)
data type coercion functions, e.g., str, int
NoneType (None)
print, input
expressions
Python Standard Library – math, turtle, random, doctest modules; import
assignment statement
Python repetition – for, while
Python conditionals – if
variable assignment
user-defined functions; function design; docstrings
IDLE interactive development environment; help function; dir, type, id

CIS 210
CIS 210 Learning Outcomes

• understand, develop, implement algorithms for computational problem solving;
• use structured design and testing methods to develop and implement programs;
• read, write, revise, document, test, and debug code;
• demonstrate robust mental models of data representation and code execution;
• demonstrate good understanding of a high level programming language;
• introduce and/or implement a sampling of classic computer science problem domains and algorithms.