Types and Variables

CS195 - Lecture 2

Instructor: Dr. V

Variables

- a variable (as opposed to a literal) can take on different values
- variables can be used in place of literal values

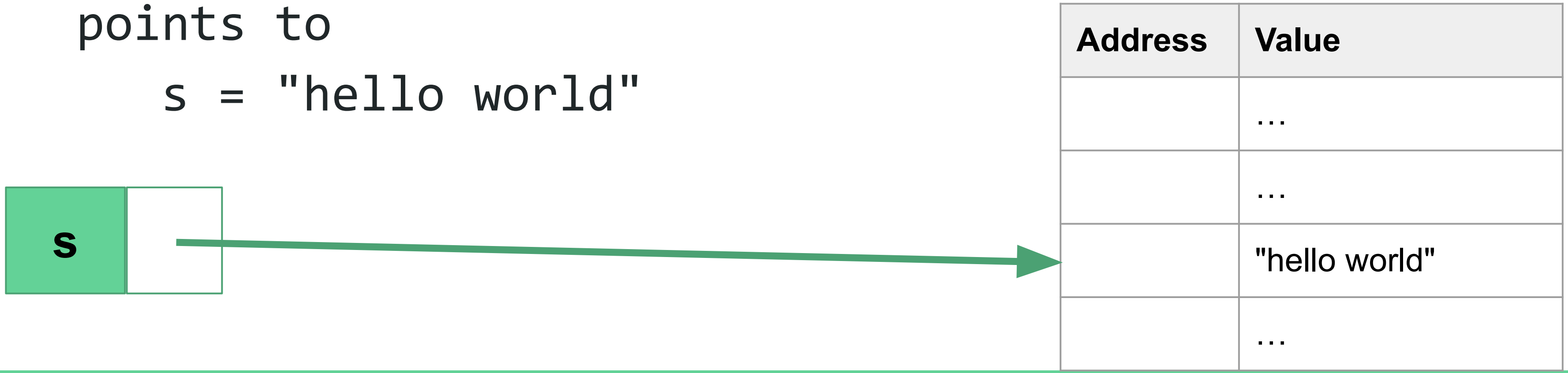
```
# print string literal "hello world"
print( "hello world" )


# assign variable s to string "hello world", then print s
s = "hello world"
print( s )
```

Variables are pointers to some place in memory

- When you are creating a variable in python, you are allocating memory on your computer for storing an address
- When you are assigning a variable to a value in python, you are changing the address this variable points to

```
s = "hello world"
```



Address	Value
	…
	…
	"hello world"
	…

Assignments

- use equal sign to assign a variable to a value

```
# what does this print?
x = "hello world"
print( x )

# what does this print?
y = 7
print( y )

# what does this print?
z = 43
print( y + z )
```

Assignments

```
# what do you think this prints?

x = 15

y = 7

x = y

z = 43

y = z

print( x + z )
```

Assignments

```
# let's trace it...

x = 15  # x:15

y = 7   # x:15    y:7

x = y   # x:7     y:7

z = 43  # x:7     y:7     z:43

y = z   # x:7     y:43    z:43

print( x + y )
```

Basic arithmetic in python

```python
# what do you think each of these statements prints?
print( 27 + 10 )

print( 27 - 10 )

print( 27 * 10 )

print( 27 / 10 )  # / is float division

print( 27 // 10 ) # // is integer division (aka floor division)

print( 27 % 10 )  # % is the modulo (aka remainder) operator

print( 27 ** 10 ) # ** is the exponent operator
```

Basic arithmetic in python - P E MD AS

```
# what do you think this prints?

x = 2 + 3 * ( 4 - 5 ) ** 6

print( x )
```

Basic arithmetic in python

```
# what do you think this prints?
x = 27 + 10

x = x + 10

x = x * 10

print( x )
```

Augmented assignments

```
# what do you think this prints?
x = 27 + 10

x += 10      # same thing as x = x + 10

x -= 7       # same thing as x = x - 7

x *= 10      # same thing as x = x * 10

print( x )
```

ZeroDivisionError

```
>>> x = 27 / 0

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Use python repl as a calculator

- type **python** in terminal, you'll enter the python repl
- you'll see >>>
- you'll get to type python statements, and will see what the result is (even without using the **print** function)

```
PS C:\...> python
Python 3.10.2…
Type "help", "copyright", "credits" or "license" for more
information.
>>> x=23
>>> x+=4
>>> x + 17 / 6
29.833333333333332
>>>
```

Input/Output

```
print( 'hello!' )
print( 'what is your name?' )

name = input( '> ' )    # assign var name to whatever user enters

print( 'hi ' + name + '! i am bot 😀' )
print( 'how old are you, ' + name + '?' )

age = input( '> ' )     # assign var age to whatever user enters

print( "oh wow. ur " + age + " years older than me 😲" )
print( 'how old are you?' )
age = input( '> ' )
print( "oh wow. ur " + age + " years older than me 😲" )

print( "if i was 10yrs old, you'd be..." )

ageDifference = age - 10
print( ageDifference + " years older than me? 🤔" )
```

```
how old are you?
> 20
oh wow. ur 20 years older than me 😲
if i was 10yrs old, you'd be...
Traceback (most recent call last):
  File "chat.py", line 7, in <module>
    ageDifference = age - 10
TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

Strongly vs Weakly Typed Programming Languages

- python is strongly typed
  - this means that variable types don't get converted automatically

```
print( 'asdf' + 'asdf' )    # ok
print( 7 + 7 )              # ok
print( 'asdf' + 7 )         # error
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

- in weakly typed languages (e.g., JavaScript)
  - variable types can get converted automatically

type

```
# what does this print?
x = "hello world"
print( x )
print( type( x ) )

# what does this print?
x = 7
print( x )
print( type( x ) )

# what does this print?
x = 7.0
print( x )
print( type( x ) )
```

Static vs Dynamically Typed Programming Languages

- python is dynamically typed (sometimes referred to as untyped)
    - this means that a variable can take on different types

      ```
      x = 'asdf'
      x = 7
      x = 7.7
      ```

- dynamically typed languages (e.g., Python, JavaScript, PHP)
    - less code, more flexible
- statically typed languages (e.g., C/C++, Java, C#, Swift)
    - faster, more memory-efficient

Converting types (aka Casting)

```
s = "7"
i = 7

# error
print( s + i )

# convert i to string before adding to s
print( s + str( i ) )

# or convert s to integer before adding to i
print( int( s ) + i )
```

Converting types (aka Casting)

```
>>> int( '7' )
7
>>> int( 7.7 )
7
>>> int( '7.7' )
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '7.7'
>>> int( 'abc' )
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'abc'
>>>
```

Converting types (aka Casting)

```
>>> float( 7 )
7.0
>>> float( '7.7' )
7.7
>>> float( 'abc' )
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: 'abc'
>>>
```

How would you change this code to get it to work?

```
print( 'how old are you?' )
age = input( '> ' )
print( "oh wow. ur " + age + " years older than me 😲" )

print( "if i was 10yrs old, you'd be..." )

ageDifference = age - 10
print( ageDifference + " years older than me? 🤔" )
```

Naming variables – Syntax and readability

```
myvariable = 7     # works
my variable = 7    # SyntaxError; var names have no whitespace

variable1 = 7      # works
1variable = 7      # SyntaxError: python vars start with A-z or _



speedoflight = 299792458     # not readable

speed_of_light = 299792458   # snake_case - readable, but lengthy

speedOfLight = 299792458     # camelCase – the best (imo)
```

Naming variables – conventions

```
myVar = 7                        # use camelCase or snake_case for vars

PI = 3.14159                     # use uppercase to denote a constant

SPEED_OF_LIGHT = 299792458    # with uppercase, use SNAKE_CASE


_hiddenVar = 7                   # start with _ to signify var is "hidden"

def myFunction(): ...             # use camelCase or snake_case for functions

class CollegeStudent: ...         # use Pascal case to denote class names
```

Do NOT use reserved keywords for naming vars

```
>>> in = 7  # in is a reserved keyword
  File "<stdin>", line 1
    in = 7
    ^^
SyntaxError: invalid syntax
```

```
>>> help("keywords")

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```

Do not overwrite another defined var/function name

```
>>> print( 'hello ')
hello



>>> print = 7

>>> print( 'hello' )
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not callable
```

Python operators

`https://www.w3schools.com/python/python_operators.asp`

Literals, variables, function names, operators

- there are 3 literals in the code below, what are they?
- there is 1 variable in the code below, what is it?
- there are 2 function calls in the code below, what are they?
- there are 3 operators in the code below (besides parentheses), what are they?

```
x = 7
print( 'abcd' + str( x ** 2 ) )
```

Multiple variable assignments

```
x = 7
y = 8
z = 9

x, y, z = 7, 8, 9
```

Variable value swapping

```
x = 7
y = 8

# in another language you'd have to do the following:
temporaryVariable = x
x = y
y = temporaryVariable

# in python you can just do this:
x, y = y, x
```

The new assignment operator :=

- Python 3.8 has added a new assignment operator :=
- You can use it inside other other statements

```
# python 3.7 and below
a = 5
print( a )


# python 3.8 and higher
print( a := 5 )
```

Assignment 2

- create a chatbot
  - asks your name
  - greets you by your name
  - asks how tall you are
  - tells you how many you's it would take to reach the moon
  - asks how much you weigh
  - tells you how much you would weigh on the moon
- make sure your code readable!
  - variable names should be meaningful
  - your code should have comments
    - comments at the top with
      - CS195 - Assignment 2 - *your name*
      - program title
    - comments throughout explaining what the code does