Chapter 2

Dr. Raza Ul Mustafa Khokhar
American University
Computer Science Department - CSC-148

Values and Data Types (1/3)

- A value is one of the fundamental things — like a word or a number — that a program manipulates.
  - The values we have seen so far are 5 (the result when we added 2 + 3), and "Hello, World!".
  - We often refer to these values as **objects** and we will use the words value and object interchangeably

→ $2+3 = 5$

Values and Data Types (2/3)

These objects are classified into different classes, or data types:

4 is an **integer**, and

"Hello, World!" is a **string**, so-called because it contains a string or sequence of letters.

You (and the interpreter) can identify strings because they are enclosed in **quotation marks**

Values and Data Types (3/3)

```
In [41]: print(type("Hello, World!"))
         print(type(17))
         print("Hello, World")
```

```
<class 'str'>
<class 'int'>
Hello, World
```

Floats & Int

- Continuing with our discussion of data types, numbers with a decimal point belong to a class called float, because these numbers are represented in a format called floating-point

```
In [42]: print(type(17))
         print(type(3.2))

         <class 'int'>
         <class 'float'>
```

Strings in python (1/2)

- Strings in Python can be enclosed in either single quotes (') or double quotes (" - the double quote character), or three of the same separate quote characters (''' or """).

```
In [43]: print(type('This is a string.') )
         print(type("And so is this.") )
         print(type("""and this.""") )
         print(type('''and even this...''') )
```

```
<class 'str'>
<class 'str'>
<class 'str'>
<class 'str'>
```

Strings in python (2/2) - Rules

- **Note**: Double quoted strings can contain single quotes inside them, as in "Bruce's beard",
- Single quoted strings can have double quotes inside them, as in 'The knights who say "Ni!"'.
- Strings enclosed with three occurrences of either quote symbol are called triple quoted strings. They can contain either single or double quotes:

Printing in python

In [46]:
```
# Comma is used to print multiple outputs
```

In [47]:
```
print(42, 17, 56, 34, 11, 4.35, 32)
print(3.4, "hello", 45)
```

```
42 17 56 34 11 4.35 32
3.4 hello 45
```

Type conversion functions

- Sometimes it is necessary to convert values from one type to another.
- The functions int, float and str will (attempt to) convert their arguments into types int, float and str respectively. We call these type conversion functions.

See example next page: Think what type of error we have ?

Type conversions example

In [48]:

```
print(3.14, int(3.14))
print(3.9999, int(3.9999))       # This doesn't round to the closest int!
print(3.0, int(3.0))
print(-3.999, int(-3.999))       # Note that the result is closer to zero

print("2345", int("2345"))       # parse a string to produce an int
print(17, int(17))               # int even works on integers
print(int("23bottles"))
```

```
3.14 3
3.9999 3
3.0 3
-3.999 -3
2345 2345
17 17

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_19048\286087422.py in <module>
      6 print("2345", int("2345"))       # parse a string to produce an int
      7 print(17, int(17))               # int even works on integers
----> 8 print(int("23bottles"))

ValueError: invalid literal for int() with base 10: '23bottles'
```

Variables

- One of the most powerful features of a programming language is the ability to manipulate variables. A variable is a name that refers to a value.

```
In [51]:
message = "What's up, Doc?"
n = 17
pi = 3.14159
```

```
In [52]:
print(message)
print(n)
print(pi)
```

```
What's up, Doc?
17
3.14159
```

```
In [53]:
print(type(message))
print(type(n))
print(type(pi))
```

```
<class 'str'>
<class 'int'>
<class 'float'>
```

Flow of variables

In [54]:

```
# What will happen here?

day = "Thursday"
print(day)
day = "Friday"
print(day)
day = 21
print(day)
```

```
Thursday
Friday
21
```

Variable names and Keywords

- Variable names can be arbitrarily long. They can contain both letters and digits, but they have to begin with a letter or an underscore.

Note: Variable names can never contain spaces.

Invalid variable names

- The underscore character (_) can also appear in a name. It is often used in names with multiple words, such as my_name or price_of_tea_in_china.
- There are some situations in which names beginning with an underscore have special meaning, so a safe rule for beginners is to start all names with a letter.

In [55]:

```
76trombones = "big parade"
more$ = 1000000
class = "Computer Science 101"
```

```
  File "C:\Users\User\AppData\Local\Temp\ipykernel_19048\3345024771.py", line 1
    76trombones = "big parade"
      ^
SyntaxError: invalid syntax
```

Python keywords – Not used as a variables

It turns out that `class` is one of the Python **keywords**. Keywords define the language's syntax rules and structure, and they cannot be used as variable names. Python has thirty-something keywords (and every now and again improvements to Python introduce or eliminate one or two):

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

Statements and Expressions

- A statement is an instruction that the Python interpreter can execute.
  - We have only seen the assignment statement so far.
  - Some other kinds of statements that we'll see shortly are while statements, for statements, if statements, and import statements.

- An expression is a combination of values, variables, operators, and calls to functions. Expressions need to be evaluated. If you ask Python to print an expression, the interpreter evaluates the expression and displays the result.

Example of expression and statement

In [57]:
```
# Simple expression
x = 5

# Arithmetic expression
result = 3 * (x + 2)
```

In [58]:
```
x = 5
# Conditional statement (if-else)
if x > 0:
    print("x is positive")
else:
    print("x is non-positive")

# Loop statement (for loop)
for i in range(5):
    print(i)

# Function definition statement
def greet(name):
    print(f"Hello, {name}!")
```

Operators and Operands

- Operators are special tokens that represent computations like addition, multiplication and division. The values the operator works on are called operands

In [59]:
```
20 + 32
hour - 1
hour * 60 + minute
minute / 60
5 ** 2
(5 + 9) * (15 - 7)
```

Operators and Operands

- The tokens +, -, and *, and the use of parentheses for grouping, mean in Python what they mean in mathematics.
- The asterisk (*) is the token for multiplication, and ** is the token for exponentiation. Addition, subtraction, multiplication, and exponentiation all do what you expect.

```
In [60]: print(2 + 3)
         print(2 - 3)
         print(2 * 3)
         print(2 ** 3)
         print(3 ** 2)
```

```
5
-1
6
8
9
```

Arithmetic Operators:

```
a = 10
b = 5

# Addition
result_add = a + b  # 10 + 5 = 15

# Subtraction
result_sub = a - b  # 10 - 5 = 5

# Multiplication
result_mul = a * b  # 10 * 5 = 50

# Division
result_div = a / b  # 10 / 5 = 2.0

# Modulus (Remainder)
result_mod = a % b  # 10 % 5 = 0

# Exponentiation
result_exp = a ** b  # 10^5 = 100000
```

Comparison Operators:

```
x = 10
y = 20

# Equal to
result_eq = x == y  # False

# Not equal to
result_neq = x != y  # True

# Greater than
result_gt = x > y  # False

# Less than
result_lt = x < y  # True

# Greater than or equal to
result_ge = x >= y  # False

# Less than or equal to
result_le = x <= y  # True
```

Logical Operators:

```
p = True
q = False

# Logical AND
result_and = p and q  # False

# Logical OR
result_or = p or q  # True

# Logical NOT
result_not_p = not p  # False
result_not_q = not q  # True
```

Assignment Operators:

```
x = 10
y = 5

# Addition assignment
x += y  # x = x + y = 15

# Subtraction assignment
x -= y  # x = x - y = 5

# Multiplication assignment
x *= y  # x = x * y = 50

# Division assignment
x /= y  # x = x / y = 2.0
```

Inputs from users

Input

In [61]:
```
n = input("Please enter your name: ")
```

```
Please enter your name:
```

In [62]:
```
n = input("Please enter your name: ")
print("Hello", n)
```

```
Please enter your name:
Hello
```

Order of Operations

- Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, **2 * (3-1)** is 4, and (1+1)**(5-2) is 8. You can also use parentheses to make an expression easier to read, as in (minute * 100) / 60, even though it doesn't change the result.

- Exponentiation has the next highest precedence, so 2**1+1 is 3 and not 4, and 3*1**3 is 3 and not 27.

- Multiplication and both division operators have the same precedence, which is higher than addition and subtraction, which also have the same precedence. So 2*3-1 yields 5 rather than 4, and 5-2*2 is 1, not 6.

- Operators with the same precedence (except for **) are evaluated from **left-to-right**. In algebra we say they are left-associative. So in the expression 6-3+2, the subtraction happens first, yielding 3. We then add 2 to get the result 5. If the operations had been evaluated from right to left, the result would have been 6-(3+2), which is 1.

Example - Order

```
In [63]: 16 - 2 * 5 // 3 + 1

Out[63]: 14
```

```
In [64]: 16 - 2 * 5 // 3 + 1
         16 - 10 // 3 + 1
         16 - 3 + 1
         13 + 1
         14

Out[64]: 14
```

floor function returns the integer value just lesser than the given rational value. ceil function returns the integer value just greater than the given rational value

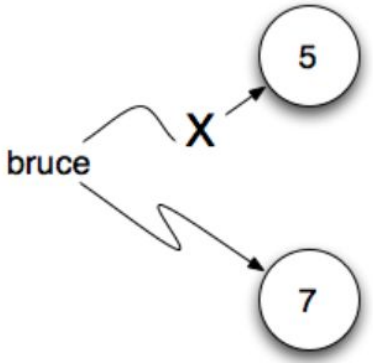
Reassignment / Updating variables

```
In [65]: bruce = 5
         print(bruce)
         bruce = 7
         print(bruce)

         5
         7
```

```
In [66]: display.Image("../pictures/reassign1.png")
```

Out[66]:




```
In [68]: x = 6
         x = x + 1

In [69]: x

Out[69]: 7
```

Thank you. Download slides & material from Canvas