Variables, Types, Expressions

Intro2CS – week 1b
Functions: reminder

```python
def petal():
    """Uses the turtle to draw a single petal""
    turtle.forward(100)
    turtle.left(120)
    turtle.forward(50)
    turtle.right(90)
    turtle.backward(50)

def flower():
    """draws a flower using the turtle.
    The flower is made up of 12 petals.
    ""
    petal();petal();petal()
    petal();petal();petal()
    petal();petal();petal()
    petal();petal();petal()
    petal();petal();petal()"
```

A multi-line comment. Use these to document all functions you write.
def a():
    print("start a()")
    b()
    c()
    print("fin a()")

def b():
    print("start b()")
    d()
    print("fin b()")

def c():
    print("start c()")
    d()
    print("fin c()")

def d():
    print("start and fin d()")

>>> a()
start a()
start b()
start and fin d()
fin b()
start c()
start and fin d()
fin c()
fin a()
Error messages and the call stack.

• When an error occurs within some function, the error message specifies the state of execution:

```python
>>> def a():
    b()

>>> def b():
    c()

>>> def c():
    d()

>>> def d():
    return "hi"*"bye"
```
```python
>>> a()
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    a()
  File "<pyshell#2>", line 2, in a
    b()
  File "<pyshell#5>", line 2, in b
    c()
  File "<pyshell#8>", line 2, in c
    d()
  File "<pyshell#11>", line 2, in d
    return "hi"*"bye"
TypeError: can't multiply sequence by non-int of type 'str'
```
Defining Functions

• Functions must be defined before they are used.

```python
a()  #ERROR: a() is not defined yet.
def a():
    # not executed until a() called:
    return b()  #Okay.

a()  #ERROR! Because b() isn’t defined.
def b():
    return 1

a()  #This is Okay.
```
Every value or object in python has a type

<table>
<thead>
<tr>
<th>“hi”, ‘hi’, “”””hi””””</th>
<th>string</th>
<th>Text. A string of characters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>-34</td>
<td>int</td>
<td>Integers ... -2,-1,0,1,2,...</td>
</tr>
<tr>
<td>0.2</td>
<td>float</td>
<td>Approximate real numbers</td>
</tr>
<tr>
<td>True, False</td>
<td>bool</td>
<td>Truth values</td>
</tr>
</tbody>
</table>

“hi”, -34, 0.2 and True are called literal values. There are many other types...
Operators

• The symbols * + / are called operators
• Use them to compose more complex expressions
• Operators work differently on different types:

```
>>> 3+3
6
>>> "hello"+"world"
'helloworld'
>>> 3.0*2
6.0
>>> "quick"*3
'quickquickquick'
>>> 20>10
True
>>> 20>"hello"
Traceback (most recent call last):
  File "<pyshell#80>", line 1, in <module>
    20>"hello"
TypeError: unorderable types: int() > str()
```
• A built in function type() can check the type for you:

```python
>>> type(3)
<class 'int'>
>>> type(3.0)
<class 'float'>
>>> type("3")
<class 'str'>
>>> |
```
List of operators

Arithmetic operators:  +  −  ⋅  /  **  %  //

(The last 3 are exponent, modulus, floor division)

Comparison operators: ==  !=  >  <  >=  <=

Logical operators: and  or  not

There are more operators, that we will cover later (e.g., assignment operators, bitwise operators)
Logical operators

<table>
<thead>
<tr>
<th>X</th>
<th>not X</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X or Y</th>
<th>X and Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
Example

What should this expression evaluate to?

\[ 3 + 4 \equiv \frac{14}{2} \text{ and not } 7 < 4.5 \]

\[ 7 \equiv \frac{14}{2} \text{ and not } 7 < 4.5 \]

\[ 7 \equiv 7 \text{ and not } 7 < 4.5 \]

\[ \text{True and not } 7 < 4.5 \]

\[ \text{True and not False} \]

\[ \text{True and True} \]

\[ \text{True} \]
Type Conversion Functions

• We can attempt to convert between types using built-in functions

```python
>>> str(12)
'12'
>>> int(12.5)
12
>>> float('12.5')
12.5
>>> bool(12)
True
>>> |
```
Variables

• We can assign values to a variable
  `<variable> = <expression>`

```python
x = 3
print(x)
y = 2+x
print(y)
```

Expression is evaluated, then result is assigned.

Variable names can be made of letters, digits, ‘_’ (but must not start with digit).

Choose meaningful names!
• Variables can be reassigned (even to different types)

```python
number = 2
print(number)
number = "hello"
print(number)
```

• The previous value is gone. Replaced by a different one.

• The output:

```
2
hello
```
• Assignments are not equations!

\[ x = 2 \]
\[ x = x + 1 \]

makes sense in Python
(but not in math)

\[ 2 + x = 5 \]

Not legal Python syntax!
(but makes sense in math)

• Note the difference between = , ==
(assignment vs. comparison operator)

What does this do?

\[ y = 3 \]
\[ x = y == 2 \]
• What is the output?

```python
x = 2
y = x
x = x+1
print(x, y)
```

17

• Does the second assignment to x change y’s value?
Another Example

#pretty print some text:
text = "hello"
length = len(text)

print("*" * (length+4))
print("*",text,"*")
print("*" * (length+4))
Input from the user

We can write a program that will accept input from its user

Using the function input([prompt])

This will pause the program and wait for input from the user. The input is returned as a string

```python
name = input("Please enter your name: ")
print("Hello", name)
```

A possible run:

```
Please enter your name: Joe
Hello Joe
```
Recap

• **Expressions** are composed of combinations of variables, operators, literals, and functions

• Expressions are evaluated by the interpreter whenever encountered.

• We’ve learned that values in Python have a type
  – We’ve seen int, string, float, bool
  – Operators work differently, depending on types

• We’ve seen two types of statements:
  • Calls to functions: `print(<expression>)`
  • Assignment: `<variable> = <expression>`
width = input("Enter rectangle's width: ")
height = input("Enter rectangle's height: ")

area = width * height
print("The area is: ", area)

Enter rectangle's width: 4
Enter rectangle's height: 3
Traceback (most recent call last):
  File "C:/Users/avivz/Desktop/tmp/expressions.py", line 4, in <module>
    area = width * height
TypeError: can't multiply sequence by non-int of type 'str'
- Actual bug found in the Harvard mark II (1947)
Correcting the problem

input() gave us strings that we later tried to multiply. Convert the input to float before doing so.

```python
width = input("Enter rectangle's width: ")
height = input("Enter rectangle's height: ")

area = float(width) * float(height)
print("The area is: ", area)
```

Enter rectangle's width: 4
Enter rectangle's height: 3
The area is: 12.0
Different errors in code

- **Intent error**: the program works, but does something we did not mean
- **Syntax error**: the interpreter stops execution since we mistyped something in the program.
  - Found just before, or during the run
- **Runtime errors**:  
  - Something unexpected crashes the program, like applying an illegal operation:

```python
>>> 5/0
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    5/0
ZeroDivisionError: division by zero
```
Other Program Examples

• Read the radius of a circle from the user, and print its area and circumference.
  – What happens if the number we receive is negative?
  – What happens if we receive a string instead?