Variables, Operators, Statements and User Input

Day 2: June 29

Our first program

```
# Our first example.
# Prints friendly greeting.

print 'Hello, WTP'
```


Our first program

```
# Our first example.
# Prints friendly greeting.

print 'Hello, WTP'
```

comments

Our first program

```
# Our first example.
# Prints friendly greeting.

print 'Hello, WTP'
```

print statement

Values (or Literals)

These are the simplest kind of expressions:

```
'Oshani'
25
True
```


Types

- Values belong to different types
- The type determines:
  - how much space in memory is needed to store the value
  - which operations can be performed on it

```
>>>type('Oshani')
<type 'str'>
>>> type( 25 )
<type 'int'>
>>>type( True )
<type 'bool'>
```

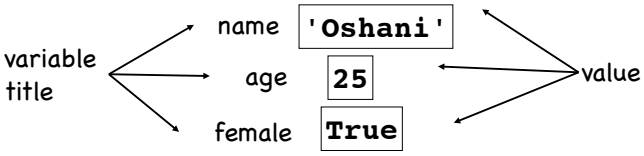
string

integer

boolean

Variables

- A variable names a location in memory.
- That location can store a value.



- A variable's value can be retrieved and altered.

age ~~25~~ 26

Variable names

- Must:
  - begin with a letter or underscore
  - contain no spaces, punctuation, or operators (but can contain underscores)
  - not be a keyword (e.g., class, if – you can find a list of Python keywords on page 11 of the text)
- Are case sensitive!

`myvariablename` ≠ `myVariableName`

Assignment Statement

- A variable can be assigned a value.

```
name = 'Jenny'
```

name **'Jenny'**

- The value of a variable can be assigned to another variable.

```
my_friend = 'Jenny'
your_friend = my_friend
```


Basic types in Python

int (integer)
  number without fractional parts (2, 73, -122)
float (floating point number)
  number with fractional parts (3.14, -18.0)
bool (boolean)
  True or False
str (string)
  sequence of characters ('WTP', 'you are the best')

Type Conversion

- int to float

```
a = float(5)
print a
```

```
5.0
```

- str to int

```
a = int('5')
print type(a)
```

```
<type 'int'>
```

- int to str

```
a = str(5)
print type(a)
```

```
<type 'str'>
```


HelloWTP, now with variables

```
1 message = 'Hello, WTP'
2 num_students = 40
3
4 print message
5 print num_students
```

What does this code output?

HelloWTP, now with variables

```
1message = 'Hello, WTP'
2print message
3
4message = 'Goodbye, WTP'
5print message
```

What does this code output?

Arithmetic operators

- Applied to numbers, produce numbers

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
%	Remainder

Arithmetic operators

- Familiar operations:

```
1 a = 5
2 b = 10
3 c = a + b
4 d = a – b
5 e = a * b
6 f = b / a
7 g = b % a
8 h = b ** 2
```


Practice with arithmetic operators

```
1 a = 19
2 b = 21
3 result = a + b
4 result = result / 10
5 result = result ** 2
6 print result
```

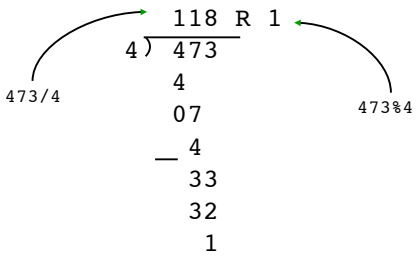
What does this code fragment output?

What About 5 / 2?

- Python performs floor division.
  - If both numbers are integers, the result is an integer.
  - Floor division chops off the fractional part of the result.
- To get a fractional result:
  - 5.0/2 = 2.5
  - float(5)/2 = 2.5

The modulus operator %

Returns the remainder when the first argument is divided by the second.



String operators

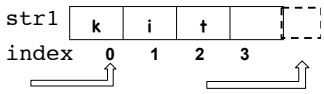
- Applied to strings, produce strings

Operator	Operation
+	Concatenation
*	Repetition
<string>[ ]	Indexing
<string>[:]	Slicing

String operators

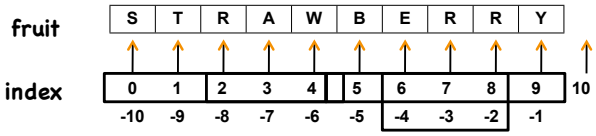
- Applied to strings, produce strings

```
1 str1 = 'kit '
2 str2 = 'kat '
3 str3 = str1 + str2          'kit kat '
4 str3 = str3 * 2       'kit kat kit kat '
5 c = str1[0]                           'k'
6 c = str1[4]  IndexError: string index out
                                  of range
```



The slicing operator [m : n]

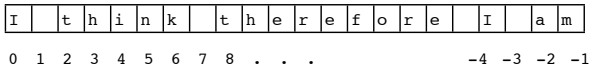
Returns the part of the string from the "m-th" character to the "n-th" character, including the first but excluding the last.




```
1 str1 = fruit[2:5]    'RAW'
2 str1 = fruit[:5]   'STRAW'
3 str1 = fruit[5:]   'BERRY'
4 str1 = fruit[6:-1]   'ERR'
```


Practice with string operators

```
1 str1 = 'I think therefore I am'
2 str2 = str1[-4:]
3 str3 = str1[7:-4]
4 print str1[2:8]*3
5 result = str2 + str3 + str1[:7]
6 print result
```



What does this code fragment output?

Relational operators

- For comparing two variables
- Type of argument depends on the operator, but always produces a bool
- Some types:

not the same as =

x == y True if x equals y
x != y True if x does not equal y
x > y True if x is greater than y
x < y True if x is less than y
x >= y True if x is greater than or equal to y
x <= y True if x is less than or equal to y

Practice with relational operators

```
1 temp = 80
2 temp_is_low = (temp < 65)
3 forecast = 'rain'
4 is_raining = (forecast == 'sunny')
5 result = (temp_is_low != is_raining)
6 print result
```

What does this code fragment output?

Boolean (logical) operators

- Applied to booleans, produce booleans
- Three types:
  - and : True only if both arguments are true
  - or : True if one or both arguments are true
  - not : True only if the argument is false

```
1 a = True
2 b = False
3 c = a and b
4 d = a or b
5 e = not a
```


Practice with boolean operators

```
1 a = False
2 b = True
3 c = False
4 result = not (b and c)
5 result = result or a
6 print result
```

What does this code fragment output?

Precedence: order of operations

- What order do these operations get evaluated in?

```
result = 5 * 9 – 42 + 17 / 3 ** 2
```

- Rules of precedence:
  - ** takes precedence over * and /, which precedes + and -
  - not takes precedence over and, which precedes or
  - with equal precedence, evaluate left-to-right

```
result = (5 * 9) - ((42 + 17) / (3 ** 2))
```

- Override rules of precedence using parentheses.

Debugging

- Programming is complicated! We all make mistakes.

  - These errors are called bugs.
  - Finding and removing bugs is called debugging.
- Three main types of bugs:
  - Syntax errors
  - Run-time errors
  - Logic (semantic) errors

Syntax errors

- Because programming languages are formal, they require perfect syntax.
- Incorrect syntax results in a syntax error.
  - SyntaxError: invalid syntax
  - SyntaxError: invalid token
- Common syntax errors
  - Keyword or invalid symbols in variable names
  - Mismatched quotation marks in strings
  - Unclosed opening operator, e.g. (or [.
  - Incorrect indentation
- IDLE will report syntax errors when you run your code!

Run-time errors

- Occur when the program has no syntax errors, but something goes wrong while it is running.
- IDLE will report run-time errors when you run your program.
- Example

```
IndexError: string index out of range
```

Other run-time errors

Variable errors:

- Slightly different variable names

```
n_students = 15
print num_students
```

NameError: name 'num_students' is not defined

Run-time errors:

- Operator doesn't apply to variable types.

```
b = 'a'
i = b + 2
```

TypeError: cannot concatenate 'str' and 'int' objects

Logic errors

- Occur when the program runs successfully, but produces the wrong output.
- The computer always does exactly what you tell it! Wrong output means you did not write the program you wanted to write.
- Also called semantic errors.
- IDLE cannot catch these!

Logic errors

- Mixing up = and ==

```
b1 = True
b2 = False
b3 = b2 = b1
```

Sets b2 to b1, rather than comparing them!

- Order of operations different than expected. When in doubt, use parentheses!

Coding style

- Your code should always be:
  - readable
  - consistent
  - commented or documented
- This applies to spacing (indentation), parentheses, name choices, etc…
- Code that is easy to read and understand is far more likely to be correct!

Style: variable names

- Should indicate purpose.

```
a = 3       # what the heck is a??
```

```
num_apples = 3 # oh, the number of apples!
```

- Should follow naming conventions.
  - start variables with a lower-case letter
  - start each "new word" in the name with underscore

```
my_variable = 7
my_variable_with_a_very_long_name = 6.177
```


Style: operators

- Use spacing and parentheses to improve readability.

```
answer=num2*7+num1/6-27*num2
```

```
answer = (num2 * 7) + (num1 / 6) - (27 * num2)
```

- Order of operations is easy to forget. Use parentheses!

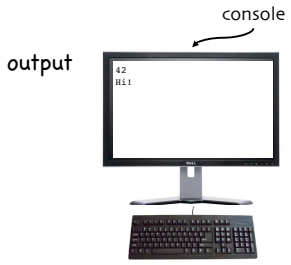
Output

- We already know how to print output to the screen.

```
print 'Hello, WTP'
```

```
num = 42
greeting = 'Hi!'
print num
print greeting
...
```

Python program

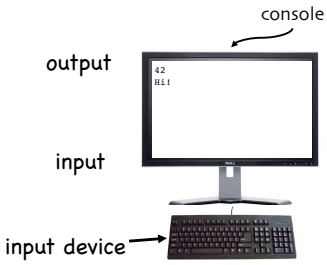


Input

- We would also like to get input from the user.

```
num = 42
greeting = 'Hi!'
print num
print greeting
...
```

Python program



User Input

- raw_input prints a prompt to the user and assigns the input to a variable as a string

```
name = raw_input('What is your name?')
```

- input can be used when we expect the input to be a number

```
age = input('How old are you?')
```


An input example

```
name = raw_input('What is your name?')
prompt = 'How old are you, ' + name + '?'
age = input(prompt)
print 'I want to be', age, 'years old too!'
```


An input example

```
name = raw_input('What is your name?')
prompt = 'How old are you, ' + name + '?'
age = input(prompt)
print 'I want to be', age, 'years old too!'
```

```
What is your name?
```


An input example

```
name = raw_input('What is your name?')
prompt = 'How old are you, ' + name + '?'
age = input(prompt)
print 'I want to be', age, 'years old too!'
```

```
What is your name?
Harry
```

An input example

```
name = raw_input('What is your name?')
prompt = 'How old are you, ' + name + '?'
age = input(prompt)
print 'I want to be', age, 'years old too!'
```

```
What is your name?
Harry
```


An input example

```
name = raw_input('What is your name?')
prompt = 'How old are you, ' + name + '?'
age = input(prompt)
print 'I want to be', age, 'years old too!'
```

```
What is your name?
Harry
How old are you, Harry?
18
```


An input example

```
name = raw_input('What is your name?')
prompt = 'How old are you, ' + name + '?'
age = input(prompt)
print 'I want to be', age, 'years old too!'
```

```
What is your name?
Harry
How old are you, Harry?
18
I want to be 18 years old too!
```


Today's exercises

- Naming and using variables
- Identifying type
- String Operators
- Boolean operators and order of operations
- User Input
- Readings for tomorrow: Sections 5.4-7, 7, 8.3, 8.5-7, 10.1-5