Data 301 Data Analytics
Python Flow Control

Dr. Irene Vrbik

University of British Columbia Okanagan
irene.vrbik@ubc.ca

Term 1, 2019

Decisions

- **Decisions** are used in programming to specify one or more conditions to be tested, along with statement(s) to execute if the condition is true.
- A **condition** is an expression that is either `True` or `False`.
- These conditions control the flow of you program and different statements will be carried out depending on the outcome of these conditions.
- To build conditional statements we need to be able to write *Boolean expressions*.

Boolean Expressions

- A **Boolean expression** is an expression that evaluates to a *Boolean value*[1] .
- A **Boolean value** is either `True` or `False`.

Boolean values

Boolean values are *not* strings. The Python type for storing `True` and `False` values is called `bool`.

```
>>> print(type(True))
<class 'bool'>
>>> print(type("True"))
<class 'str'>
```

[1]The name comes from George Boole, who first defined an algebraic system of logic in the mid 19th century

Boolean Expressions

We can create Boolean expressions using:

Relational operators/Comparisons: used to compare two values

- Examples:

```
5 < 10  # returns True
N > 5   # N is a variable. Answer depends on N.
```

Logical operators: the logical operators `and`, `or` and `not` are used to combine relational operators.

- Example: `(n > 5) and (v != n)`

The result these expressions are a *Boolean value* which is either `True` or `False`.

Comparisons

A **condition** is a Boolean expression that is either `True` or `False` and may contain one or more comparisons.

The comparison operators in Python are summarized below:

Syntax	Description
`>`	Greater than
`>=`	Greater than or equal
`<`	Less than
`<=`	Less than or equal
`==`	Equal (Note: Not "=" which is used for assignment!)
`!=`	Not equal

Conditions with `and`, `or`, `not`

Conditions may be combined using the relational operators `and`, `or`, `not`.

True if:	Syntax	Examples	Outpu
both are true	`and`	`True and True`	`True`
		`False and True`	`False`
either or both are T	`or`	`True or True`	`True`
		`False or True`	`True`
		`False or False`	`False`
false	`not`	`not True`	`False`
		`not False`	`True`

Condition Examples

```
n = 5
v = 8
print(n > 5) #False
print(n == v) #False
print(n != v) # True
print((n == v) and (n+4>v)) # False
print((n == v) or (n+4>v)) # True
print((n+1) == (v-2) or (not v>4)) # True
print((n+1) == (v-2) or not v>4) and (n>5)) # False
```

Order of Operations

Table: The order of operations; see complete list here.

`()`	brackets
`**`	exponents
`* / % MOD`	Multiplication, division, modulo
`+ -`	Addition and subtraction
`< <= > >=`	Comparisons: less-than and greater-than
`== !=`	Comparisons: equal and not equal
`and`	and
`or`	or
`not`s always bind to the condition immediate next to it	

Tip:

I recommend always using brackets to avoid confusion.

Example 16

How many of the following conditions are TRUE?

1. True and False
2. not True or not False
3. `3 + 2 == 5 or 5 > 3 and 4 != 4`
4. `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
5. `not (True or False) or True and (not False)`

A) 1 B) 2 C) 3 D) 4 E) 5

Answer:

How many of the following conditions are TRUE?

1. True and False
2. not True or not False = `(not True) or (not False)`
3. `3 + 2 == 5 or 5 > 3 and 4 != 4`
   `= (5 == 5) or (5 > 3) and (4 != 4)`
   `= (5 == 5) or ((5 > 3) and (4 != 4))` # and first
4. `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
   `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
   `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
5. `not (True or False) or True and (not False)`
   `not (True or False) or True and (not False)`
   `not (True or False) or True and (not False)`

A) 1 B) 2 C) 3 D) 4 E) 5

Answer:

How many of the following conditions are TRUE?

1. True and False
2. not True or not False = `(not True) or (not False)`
3. `3 + 2 == 5 or 5 > 3 and 4 != 4`
   `= (5 == 5) or (5 > 3) and (4 != 4)`
   `= (5 == 5) or ((5 > 3) and (4 != 4))` # and first
4. `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
   `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
   `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
5. `not (True or False) or True and (not False)`
   `not (True or False) or True and (not False)`
   `not (True or False) or True and (not False)`

A) 1 B) 2 C) 3 D) 4 E) 5

Answer:

How many of the following conditions are TRUE?

1. True and False
2. not True or not False = `(not True) or (not False)`
3. `3 + 2 == 5 or 5 > 3 and 4 != 4`
   `= (5 == 5) or (5 > 3) and (4 != 4)`
   `= (5 == 5) or ((5 > 3) and (4 != 4))` # and first
4. `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
   `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
   `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
5. `not (True or False) or True and (not False)`
   `not (True or False) or True and (not False)`
   `not (True or False) or True and (not False)`

A) 1 B) 2 C) 3 D) 4 E) 5

Answer:

How many of the following conditions are TRUE?

1. True and False
2. not True or not False = `(not True) or (not False)`
3. `3 + 2 == 5 or 5 > 3 and 4 != 4`
   `= (5 == 5) or (5 > 3) and (4 != 4)`
   `= (5 == 5) or ((5 > 3) and (4 != 4))` # and first
4. `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
   `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
   `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
5. `not (True or False) or True and (not False)`
   `not (True or False) or True and (not False)`
   `not (True or False) or True and (not False)`

A) 1 B) 2 C) 3 D) 4 E) 5

Answer:

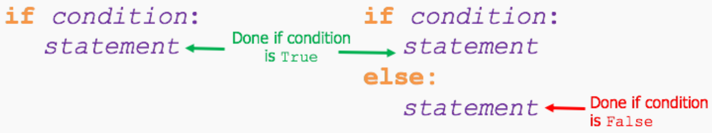
How many of the following conditions are TRUE?

1. True and False
2. not True or not False = `(not True) or (not False)`
3. `3 + 2 == 5 or 5 > 3 and 4 != 4`
   `= (5 == 5) or (5 > 3) and (4 != 4)`
   `= (5 == 5) or ((5 > 3) and (4 != 4))` # and first
4. `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
   `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
   `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
5. `not (True or False) or True and (not False)`
   `not (True or False) or True and (not False)`
   `not (True or False) or True and (not False)`

A) 1 B) 2 C) 3 [✓ D) 4] E) 5

Decisions

In Python decision syntax:



```
if condition:
    statement
```

```
if condition:
    statement
else:
    statement
```

- The statement(s) after the `if` condition is only performed if the *condition* (i.e. Boolean expression) returns `True`.
- Any statement(s) following the (optional) `else:` condition is only performed if the *condition* is `False`.

Python syntax

Remember that the indentation and colons are *not* optional!

Decision Block Syntax

- Statements listed after an `if/elif/esle` clause are not only indented for readability.
- These indentation is also how Python knows which statements are part of the group of statements to be executed.
- Statements with the same indentation belong to the same group called a **suite**.
- Be consistent with either using tabs or spaces (no mixing)

Tip: one-line if clause

If the suite of an if clause consists of a single line, it may go on the same line as the header statement.

```
if (n > 100): print("n is large")
```

Decisions if/elif Syntax

Check out the difference for `age = 20`:

```
age = 20
if age > 19:
    print("Not a teenager")
    print("Sorry")
else:
    print("You're young")
    print("ID checked")
```

The above returns:

```
Not a teenager
Sorry
```

```
age = 20
if age > 19:
    print("Not a teenager")
    print("Sorry")
else:
    print("You're young")
print("ID checked")
```

The above returns:

```
Not a teenager
Sorry
ID checked
```

Generic code:

```
if (cond1):
    Process 1
```

Example 1:

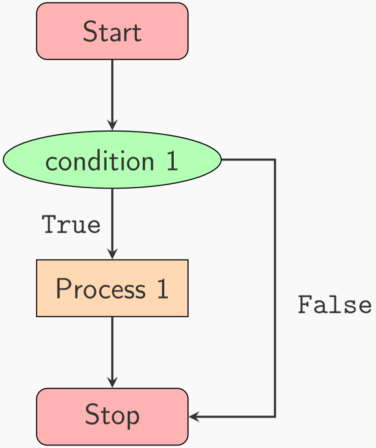
```
n = 5
if (n < 10):
    n = 10
```

`n` is now 10

Example 2:

```
n = 5
if (n > 10):
    n = 10
```

`n` remains 5



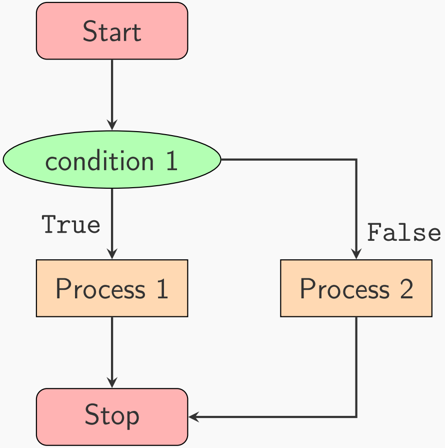
Generic code:

```
if (cond1):
    Process 1
else:
    Process 2
```

Example 3:

```
n = 5
if (n > 10):
    n = 10
else:
    n = 3
```

`n` is now 3



Decisions if/elif Syntax

If there are more than two choices, use `if/elif/else` statements.
N.B. once a condition is met, no subsequent conditions are checked

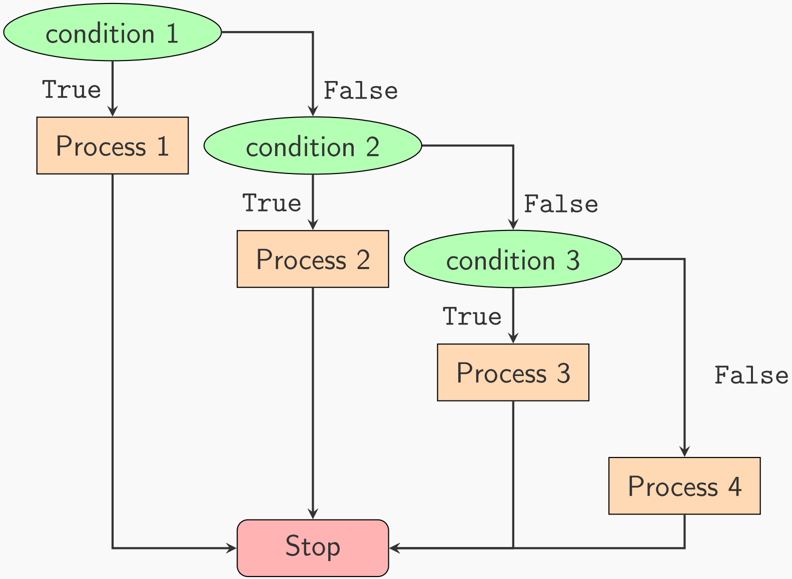
```
if condition1:
    Process 1
elif condition2:
    Process 2
elif condition3:
    Process 3
else:
    Process 4
```

```
if n == 1:
    print("one")
elif n == 2:
    print("two")
elif n == 3:
    print("three")
else:
    print("Too big!")
print("Done!")
```

`else`

Again, the `else` statement is an optional. There could be at most one `else` statement following an `if`.

`if, elif, else`


condition 1
True
False
Process 1
condition 2
True
False
Process 2
condition 3
True
Process 3
False
Process 4
Stop

Decisions if/elif Syntax

```
1  n = 1
2  if n == 1:
3      print("one")
4  elif n>0: # this condition is never checked since the
5      # condition on line 2 has already been satisfied
6      print("positive number")
7  elif n == 3:
8      print("three")
9  else:
10     print("Too big!")
11 print("and Done!") #  not part of the if statement
```

The above returns:

```
one
and Done!
```

Decisions if/elif Syntax

```
1  n = 3
2  if n == 1:
3      print("one")
4  elif n>0:
5      print("positive number")
6  elif n == 3: # this condition is never checked since
7      # condition on line 4 has already been satisfied
8      print("three")
9  else:
10     print("Too big!")
11 print("and Done!") #  not part of the if statement
```

The above returns:

```
positive number
and Done!
```

Decisions Multiple `if` statements

- As mentioned previously, once a condition is met in an if/elif statement, no subsequent conditions are checked.
- If we want all conditions to be checked we could use multiple `if` statements:

```
if condition1:
    Process 1
if condition2:
    Process 2
if condition3:
    Process 3
if condition4:
    Process 4
```

```
n = 3
if n > 0:
    print("positive number")
if n == 3:
    print("three")
if n < 10:
    print("single digit")
```

↑ returns →

```
positive number
three
single digit
```

Example 17

What is the output of the following code?

```
n = 3
if n < 1:
    print("one")
elif n > 2:
    print("two")
elif n == 3:
    print("three")
```

A) nothing

B) one

C) two

D) three

E) error

Answer:

What is the output of the following code?

```
n = 3
if n < 1:
    print("one")
elif n > 2:
    print("two")
elif n == 3:
    print("three")
```

A) nothing

B) one

C) two

D) three

E) error

Example 18

What is the output of the following code?

```
n = 3
if n < 1:
    print("one")
elif n > 2
    print("two")
else:
    print("three")
```

A) nothing

B) one

C) two

D) three

E) error

Answer:

What is the output of the following code?

```
n = 3
if n < 1:
    print("one")
elif n > 2
    print("two")
else:
    print("three")
```

1. nothing
2. one
3. two
4. three
5. error (missing colon)

Example 19

What is the output of the following code?

```
n = 1
if n < 1:
    print("one")
elif n > 2:
    print("two")
else:
    print("three")
print("four")
```

A) nothing **B)** one four **C)** three **D)** three four **E)** error

Answer:

What is the output of the following code?

```
n = 1
if n < 1:
    print("one")
elif n > 2:
    print("two")
else:
    print("three")
print("four")
```

A) nothing B) one four C) three D) *three four* E) error

Example 20

What is the output of the following code?

```
n = 0
if n < 1:
    print("one")
    print("five")
elif n == 0:
    print("zero")
else:
    print("three")
print("four")
```

A) nothing

B) one
four

C) one
five
four

D) one
five
zero
four

E) error

Answer:

What is the output of the following code?

```
n = 0
if n < 1:
    print("one")
    print("five")
elif n == 0:
    print("zero")
else:
    print("three")
print("four")
```

A) nothing

B) one
four

C) *one*
five
four

D) one
five
zero
four

E) error

Try it: Decisions

Example 21

Write a Python program that asks the user for a number then prints out if it is even or odd.

Example 22

Write a Python program that asks the user for an integer. If that number is between 1 and 5, prints out the word for that number (e.g. 1 is one). If the number is not in that range, print out error.

Loops and Iteration

A **loop** repeats a set of statements multiple times until some condition is satisfied.

- Each time a loop is executed is called an **iteration**.

A `for` loop repeats statements a certain number of times.

- It will iterate over a sequence, eg. 1, 2, . . . 10
- or it could iterate over group/collection elements, eg. lines in a document, elements in a list

A `while` loop repeats statements while a condition is True.

- At each iteration we will check this condition.
- If its `True` we complete another iteration
- If its `False` we exit the loop.

`while` loops

The most basic looping structure is the *while* loop.

A while loop continually executes a set of statements while a condition is true. Syntax:

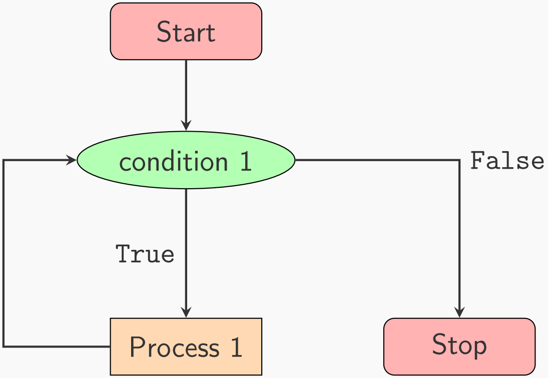
```
while condition :
    statement1
    statement2
        ⋮
```

Example:

```
n = 1
while n <= 5:
    print(n)
    n = n + 1
```

prints the values 1 through 5.

`while` loops



Shorthand

In addition to the = operator for assigning a value to a variable, Python also supports a shorthand version that compounds various mathematical operators with the assignment operator:

Table: Table taken from this source

Operator	Example	Equivalent to
`=`	`x = 5`	`x = 5`
`+=`	`x += 5`	`x = x + 5`
`-=`	`x -= 5`	`x = x - 5`
`*=`	`x *= 5`	`x = x * 5`
`/=`	`x /= 5`	`x = x / 5`
`%=`	`x %= 5`	`x = x % 5`

Hence the program from 2 slides ago can be written:

```
n = 1
while n <= 5:
    print(n)
    n += 1
```

Output:

```
1
2
3
4
5
```

Question: `while` loop

Example 23

What is the output of the following code:

```
n = 4
while n >= 0:
    n = n - 1
    print(n)
```

A) numbers 3 to -1

B) numbers 3 to 0

C) numbers 4 to 0

D) numbers 4 to -1

E) numbers 4 to infinity

Question: `while` loop

answer

What is the output of the following code:

```
n = 4
while n >= 0:
    n = n - 1
    print(n)
```

A) *numbers 3 to -1*

B) numbers 3 to 0

C) numbers 4 to 0

D) numbers 4 to -1

E) numbers 4 to infinity

Question: `while` loop 2

Example 24

What is the output of the following code:

```
n = 1
while n <= 5:
    print(n)
n = n + 1
```

A) nothing

B) numbers 1 to 5

C) numbers 1 to 6

D) lots of 1s

Question: `while` loop

answer

What is the output of the following code:

```
n = 1
while n <= 5:
    print(n)
n = n + 1
```

A) nothing

B) numbers 1 to 5

C) numbers 1 to 6

D) *lots of 1s* *Infinite loop without the fourth line indented*

The `for` loop

- A `for` loop repeats statements a given number of times.
- One way of building a for loop is to iterate over a sequence which we create using `range()`

```
for i in range(1,6):
    print(i)
```

- The above prints the numbers 1 through **5**.

`range(start, end)`

In `range(start, end)`, the `start` number in inclusive and the `start` number is *exclusive*.

Using `range()`

- The general form of range is:

  `range(start, end, step)`
- The default `step` (i.e increment) is 1
- We may also specify an increment:

```
# prints the numbers: 1,3,5,7,9
for i in range(1, 10, 2):
    print(i)
# prints the numbers: 2,4,6,8
for i in range(2, 10, 2):
    print(i)
# prints the numbers 5 to 1
for i in range(5,0, -1):
    print(i)
```

Using `range()`

- It is only required that the `end` argument be provided for the `range()` function.
- If the `start` argument is not provided, it is set as its default value of **0** (not 1).

```
for i in range(4):
    print(i)
```

The above prints the numbers: `0,1,2,3` (remember, `end` is *not* inclusive)

the for and while loop

The `for` loop is like a short-hand for the `while` loop:

-
  ```
  i=0
  while i < 10:
      print(i)
      i += 1
  ```

-
  ```
  for i in range(0, 10, 1):
      print(i)
  ```

Common problems – Infinite Loops

Infinite loops are caused by an incorrect loop condition or not updating values within the loop so that the loop condition will eventually be false.

- Example:

```
n = 1
while n <= 5:
    print(n)
```

Here we forgot to increase n $\rightarrow$infinite loop.

N.B. to exit from an infinite loop while running Python in the console, press Ctrl + C (press the stop icon in Jupyter Notebook).

Common Problems – Off-by-one Error

The most common error is to be "off-by-one". This occurs when you stop the loop one iteration too early or too late.

- Example:

```
for i in range(0,10):
    print(i)
```

  This loop was supposed to print 0 to 10, but it does not.

Example 25

Question: How can we fix this code to print 0 to 10?

Question: `for` loop

Example 26

How many numbers are printed in this loop

```
for i in range(1,10):
    print(i)
```

A) 0

B) 9

C) 10

D) 11

E) error

Answer:

How many numbers are printed in this loop

```
for i in range(1,10):
    print(i)
```

A) 0

B) *9*

C) 10

D) 11

E) error

Question: `for` loop

Example 27

How many numbers are printed in this loop

```
for i in range(11,0):
    print(i)
```

A) 0

B) 9

C) 10

D) 11

E) error

Answer:

How many numbers are printed in this loop

```
    for i in range(11,0):
        print(i)
```

A) *0*

B) 9

C) 10

D) 11

E) error

Try it: for loops

Example 28

Write a program that prints the numbers from 1 to 10 then 10 to 1.

Example 29

Write a program that prints the numbers from 1 to 100 that are divisible by 3 and 5.

Example 30

Write a program that asks the user for 5 numbers and prints the maximum, sum, and average of the numbers.

Functions and Procedures

A **procedure** is a sequence of program statements that have a specific task that they perform.

A **function** is a procedure that returns a value after it is executed.

Loosely speaking, functions are a special type of procedure for which we do not immediately know the result.

A procedure is a set of command which can be executed in order. A function returns a value and a procedure just executes commands.

While there are many built in functions at our disposal in Python, we can also create own *user-defined functions*.

Defining and Calling Functions and Procedures

Creating a function involves writing the statements and providing a function declaration with:

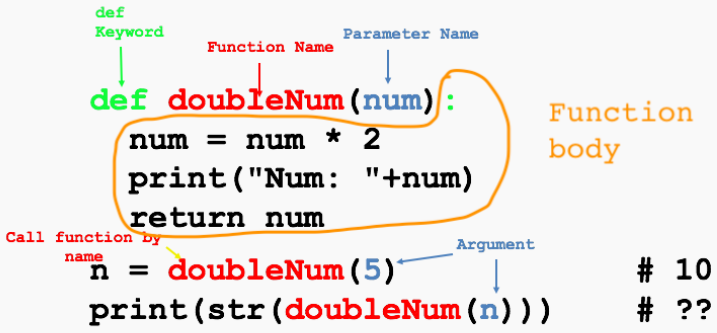
- a name (follows the same naming rules as variables)
- list of the inputs (called parameters)
- the output (return value) if any

Calling (or executing) a function involves:

- providing the name of the function
- providing the values for all arguments (inputs) if any
- providing space (variable name) to store the output (if any)

Defining and Calling a Function

Consider a function that returns a number doubled:



```
def doubleNum(num):
  num = num * 2
  print("Num: "+num)
  return num
n = doubleNum(5)                  # 10
print(str(doubleNum(n)))          # ??
```

Defining and Calling a Function

- Function “blocks"[2] begin with the keyword `def` (short for define) followed by the function name.
- Regardless of whether or not the function has any parameters, we need to follow the function name with parentheses `()`
  - Inside the parentheses, separate as many parameters as you need by commas (no parameters should have the same name).
  - A function may have 0 parameter inputs.
- The code block within every function starts with a colon `:`
- The statements that form the body of the function starts from the next line of function definition and must be indented.

[2]A block is a piece of Python program text that is executed as a unit.

Functions and Procedures

See this procedure called `hi` that prints out `Hi!`

```
def hi():
    print("Hi!")
```

Calling this procedure twice (we know exactly what to expect each time):

```
>>> hi()
hi!
>>> hi()
hi!
```

See this function called `addf` which adds two numbers (or concatenates two strings)

```
def addf(x, y):
    return x + y
```

Calling the function with integers vs. strings:

```
>>> addf(2,5)
7
>>> addf("2","5")
'25'
```

Defining and Calling a Function

- Function bodies can contain one or more `return` statement.
- The `return` statement exits a function and returns the value of the expression following the keyword.
- A function without an explicit `return` statement returns `None` (usually suppressed by the interpreter).
- Example:

```
def plus2(x):
    x + 2
```

  Since we didn't specify a `return` statement, the calculation is not provided as output.

```
>>> plus2(3)
>>> nothing = plus2(3)
>>> print(nothing)
None
```

Defining and Calling a Function

- A function can return exactly one *object*.
- If we want to return multiple values, we can return a list or a tuple, for example.

The following returns `x` and `x + 2`

```
# returning multiple values in a function using a list
def plus2(x):
    out = x + 2
    return [x,out]

# returning multiple values in a function using a tuple
def plus2(x):
    out = x + 2
    return (x,out)
```

Functions and Procedures

```
def gradeLetter(pgrade):
  if (pgrade >= 80):
     return "A"
  elif (pgrade >= 68):
    return "B"
  elif (pgrade >= 55):
    return "C"
  elif (pgrade >= 50):
    return "D"
  else:
    return "F"
```

```
>>> gradeLetter(81)
'A'
>>> gradeLetter(45)
'F'
```

```
def grade(pgrade):
  if (pgrade >= 80):
    grade = "A"
  elif (pgrade >= 68):
    grade = "B"
  elif (pgrade >= 55):
    grade = "C"
  elif (pgrade >= 50):
    grade = "D"
  else:
    grade = "F"
  return [grade, pgrade]
```

```
>>> grade(81)
['A', 81]
```

unctions and Procedures

We will often save our return value(s) to an object defined within the function to be returned.

```
def testfun(x,y,z):
    out = x+y/z
    return out
```

Notice that the variables we define within our functions will not be defined outside of that function.

```
>>> testfun(3,8,4)
5.0
>>> out
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'out' is not defined
```

Python Built-in Math Functions

Last class we had to calculate the max and average value of 5 numbers inputted by the user. There are many useful mathematical functions available in the `math` module that can help us with such calculations.

```
# Math
import math
print(math.sqrt(25))

# Import only a function
from math import sqrt
print(sqrt(25))

# Print all math functions
print(dir(math))
```

Other Python Built-in Functions

- `max, min, abs:`

  ```
  print(max(3, 5, 2)) # 5
  print(min(3, 5, 2)) # 2
  print(abs(-4)) # 4
  ```

- `type()` returns the argument data type:

  ```
  print(type(42)) # <class 'int'>
  print(type(4.2)) # <class 'float'>
  print(type('spam')) # <class 'str'>
  ```

Python Random Numbers

- Use random numbers to make the program have different behaviour when it runs.

```
from random import randint
coin = randint(0, 1) # 0 or 1
die = randint(1, 6) # 1 to 6
print(coin)
print(die)
```

Advanced: Python Functions

Python supports functional programming allowing functions to be passed like variables to other functions.

- Lambda functions are functions that do not have a name.
- Example:

```
def doFunc(func, val):
    return func(val)

print(doFunc(doubleNum, 10)) # 20
print(doFunc(lambda x: x * 3, 5)) # 15
```

Example 31

What is the value printed:

```
def triple(num):
    return num * 3

n = 5
print(triple(n)+triple(2))
```

A) 0 **B)** 6 **C)** 15 **D)** 21 **E)** error

Answer:

What is the value printed:

```
def triple(num):
    return num * 3


n = 5
print(triple(n)+triple(2))
```

A) 0 B) 6 C) 15 D) *21* E) error

Practice Questions: Functions

Example 32

1) Write a function that returns the largest of two numbers.

2) Write a function that prints the numbers from 1 to N where N is its input parameter.

Call your functions several times to test that they work.

Conclusion

Python is a general, high-level programming language designed for code readability and simplicity.

Programming concepts covered:

- variables, assignment, expressions, strings, string functions
- making decisions with conditions and if/elif/else
- repeating statements (loops) using for and while loops
- reading input with input() and printing with print()
- data structures including lists and dictionaries
- creating and calling functions, using built-in functions (math, random)

Python is a powerful tool for data analysis and automation.

Objectives

- Explain what is Python and note the difference between Python 2 and 3
- Define: algorithm, program, language, programming
- Follow Python basic syntax rules including indentation
- Define and use variables and assignment
- Apply Python variable naming rules
- Perform math expressions and understand operator precedence
- Use strings, character indexing, string functions
- String functions: split, substr, concatenation
- Use Python datetime and clock functions
- Read input from standard input (keyboard)

Objective (cont'd)

- Create comparisons and use them for decisions with if
- Combine conditions with and, or, not
- Use if/elif/else syntax
- Looping with for and while
- Create and use lists and list functions
- Advanced: list comprehensions, list slicing
- Create and use dictionaries
- Create and use Python functions
- Use built-in functions in math library
- Create random numbers
- Advanced: passing functions, lambda functions